30 Hour BASIC

CLIVE PRIGMORE

BBC
NEC

ORIC EDITION

THE COMPUTER PROGRAMME BBC TV
NATIONAL EXTENSION COLLEGE

National Extension College

30 Hour BASIC

Oric Version

by
Clive Prigmore
(Orpington College of Further Education)

Adapted for the Oric by Paul Shreeve

National Extension College Correspondence Texts

Course Number: M270

ISBN 0 86082 395 4

Designer: Peter Hall
Photographs: Michael Manni, Cambridge.
Cover photograph: Reeve photography, Cambridge.
Printed by NEC Print.
Cover printed by Burlington Press, Foxton, Cambridgeshire.

First printing 1983

Contents

How to use this course

Aims of this course

Quite simply, to help you learn to use a microcomputer with confidence. To do that you need to master three things: (a) the BASIC language, as implemented on the Oric microcomputer; (b) planning good program structures; and (c) using the Oric keyboard. This course mainly teaches you the first two.

30 Hour BASIC isn't all there is to know about BASIC but it does cover all the essentials. Once you've completed the course, you will be ready to use a textbook on BASIC or to try the second stage BASIC course which will shortly be available from NEC.

Do I need a microcomputer?

You can do the course whether or not you have access to a microcomputer, although it will be of more use to you if you can use an Oric, since the course material is designed particularly to apply to that machine.

All you have to do is to choose one of the following ways of working through the course:

Self-instructional use: With an Oric microcomputer: do all the exercises and self-assessment questions (SAQs) and key in all the programs marked $\boxed{K}$. Without a microcomputer: do all the exercises and SAQs but omit the items marked $\boxed{K}$.

FlexiStudy use: With your own Oric computer: do all the exercises and SAQs at home and key in at home all the programs marked $\boxed{K}$. Test your assignment answers on your own microcomputer. Then take/send your problems to your local FlexiStudy centre. Without your own microcomputer: do all the exercises and SAQs but ignore the items marked $\boxed{K}$. Then do the assignment questions and take these to your local FlexiStudy centre to run on their Orics.

NEC correspondence students: With your own Oric microcomputer: do all the exercises and SAQs at home and key in all the items marked $\boxed{K}$. Test your assignments on your own Oric before sending them to your NEC correspondence tutor. If the assignment programs don't run properly, tell your tutor what response you are getting from your microcomputer. Without your own microcomputer: do all the exercises and SAQs at home but ignore the items marked $\boxed{K}$. Do the assignment questions and post these to your NEC tutor.

Structure of the course

The course is in nine units (see Contents). Each unit includes:

Examples: These are problems which we solve completely for you in the text.

Self Assessment Questions (SAQs): We ask you to stop and quickly check that you have understood a new idea that we have introduced. Answers to these always appear at the end of the unit in which the SAQ occurs.

Exercises: These are longer problems for you to try. Answers appear at the end of the unit in which the exercise occurs.

$\boxed{K}$ stands for key. This is where we think you could find it helpful to key a program into your own microcomputer.

Assignments: These are questions for you to answer and send to your tutor for marking and comment. There are no answers to these in this course text.

UNIT 1
Simple statements and commands

1.1 What does a computer do?

In broad terms, a computer is a machine which helps us to solve certain kinds of problems. These usually involve symbols or characters which are familiar to us through everyday use, e.g. letters of the alphabet (capital and lower case), numbers, punctuation marks and some special characters such as +, #, ★. The computer allows us to put in one set of symbols or characters and get out a different but related set. If this seems very vague and too general, let's consider some specific examples.

CHARACTERS IN	CHARACTERS OUT
Numbers representing the size of a window.	Cost of double glazing.
List of books borrowed from a library.	List of those books overdue.
A person's name.	The person's telephone number.
Standard notation for a move in a game of chess.	Picture of the chess board with the move accomplished.
Number representing height and acceleration.	Picture of lunar lander.
Pre-determined codes.	Musical sounds.

Figure 1 Some uses of a computer

This course is not about how the computer does these things but about how you can get it to do them by giving it the right instructions. We shan't therefore be going into any detail of the insides of a computer but you will find it helpful to know which are the major parts of a computer. This is quickly dealt with in the next section.

1.2 What is a computer?

A simple model of a computer is shown in Figure 2.

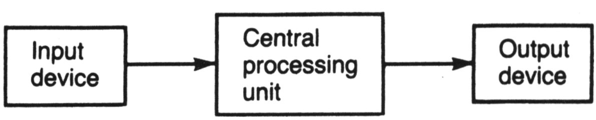


Figure 2 A simple model of a computer

You can see that there are three main parts to a computer:

1 The input device which allows you to enter either instructions or data (information) into the computer. On a microcomputer the input device is a keyboard which looks like a typewriter.

2 The central processing unit (CPU) which, amongst other things, carries out the instructions you have put in. This processing results in a modification of your data giving you the 'answer' or output that you require.
3 An output device which enables you to receive the result of the processing. The output device might be a television screen displaying the output or a printer which actually prints the output onto paper.

All this may sound very mundane. Indeed it would be were it not for the three key characteristics of a computer: (a) its capacity to store very large quantities of data which (b) it is able to process very rapidly and (c) its capacity to store a program which controls its own operation. This last characteristic is by far the most important one and is the one we are going to cover in this course.

Backing store

We shall just mention one other technical detail before looking at programming. If you are using this course you are likely to own or use a microcomputer with a small internal storage capacity. It uses that storage to keep its main running instructions plus the details of the problem it is currently solving. The latter details are erased when you switch the machine off so, if you want to keep your program or data, you have to keep them in a backing store: a separate storage system that you can link to the computer as needed. On small systems this will be an ordinary audio cassette tape and on larger systems, a magnetic disc store.

So, to summarise, the main elements of a computer system are illustrated in Figure 3.

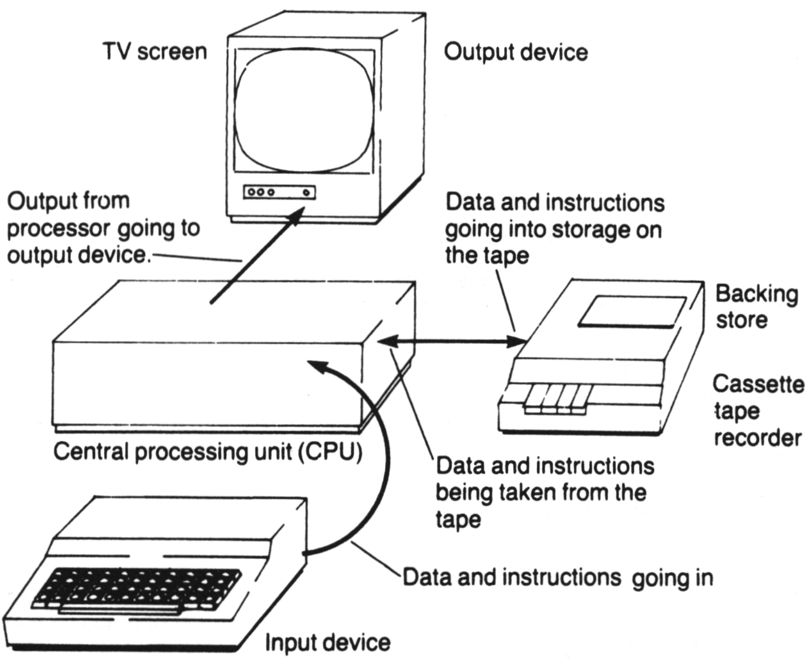


Figure 3 A typical computer system

1.3 What is BASIC?

A computer is an electronic device which processes patterns of electrical signals. If you had a problem, you wouldn't be able to feed it into the computer as electrical signals. Nor, if the computer was ready to give you an output, would you be able to understand it if it came as electrical signals. So the computer has a machine code inside it (put there by the manufacturer) to enable it to understand a programming code that you can understand. Machine codes are called low-level programming languages and correspond directly with the patterns of electrical signals in the computer. For obvious reasons, this program is called an interpreter. This course teaches you BASIC which is a high-level programming language. You will then be able to use BASIC to program any computer that contains a BASIC interpreter. BASIC, by the way, stands for Beginners' All-purpose Symbolic Instruction Code.

You may find it useful to note the sequence of events that is taking place when you program a computer.

1 You have a problem.
2 You break down the problem into steps which can be put into BASIC.
3 You write the program in BASIC.
4 You sit at a keyboard and enter your program into the computer.
5 The computer interprets your BASIC instructions into its own code and processes them.
6 The computer prints out the results in the form you specified in the program.

And that is all you need to know about what a computer is. From now on we will assume that all you want to do is to give the computer problems and to get back results so now let's move on to a simple problem which we might want to give to a computer.

1.4 A simple problem

The main activity of programming is breaking down the solution to a problem into simple steps which can be represented by BASIC programming statements.

Imagine that you are playing the part of a computer with a young child. The child might give you two numbers and ask you to tell him their sum. After a short time the child will naturally try you out with large numbers which you cannot add in your head, so you will have to have a paper and pencil at hand. The following could be a typical dialogue.

CHILD: 'Start'
YOU: 'Give me the first number'
CHILD: '12157'
(You write this number on a piece of paper)
YOU: 'Give me the second number'
CHILD: '7896'
(You write the second number on the piece of paper)
(You perform the addition sum)
YOU: '20053'

We could describe the computer's part in this process more formally in the following way:

1 Input the first number
2 Input the second number
3 Add the two numbers
4 Output the result

Figure 4 Computer processes in adding two numbers

By this simple analogy we have arrived at a strategy for solving this problem. Broadly speaking, phases 1 and 2 would be concerned with entering numbers into the computer, phase 3 would involve a process in the central processing unit, while phase 4 would involve the output device.

Now, although we have not taught you any BASIC programming yet, we are going to show you what the problem solving sequence would look like when written in BASIC.

Example 1
Write a BASIC program to enter two numbers into the computer and to output their sum.

Solution
We have already worked out an intuitive procedure to solve this problem in Figure 4. A program in BASIC would have the following form:

```
10 INPUT FS
20 INPUT SC
30 LET SUM = FS + SC
40 PRINT SUM
50 END
```

Program 1

END is not needed on all computers – e.g. it is not mandatory in Oric BASIC.

We do not wish to concentrate on the details of the program at this stage, but hope that you can see, without stretching the imagination too far, how the strategy from Figure 4 has been changed into a program. A program then is a 'sequence of instructions composed for solving a given problem by computer'.
FS stands for First
SC stands for Second

We have to choose these odd abbreviations because the Oric-1 cannot accept any store names which include words from the BASIC language; ON is such a word, and SECOND includes the word ON; it is thus unaceptable to the Oric as a store name, and, if used, would generate an error code:

```
? SYNTAX ERROR
```

This is why we have not used the more meaningful FIRST and SECOND. Don't worry if this seems obscure at present—more details will be suppied as the course text proceeds.

SAQ 1
We now come to the first point in the course at which we want you to check your progress through trying this Self-Assessment Question (SAQ). The SAQs are designed to help you find out whether or not you have understood the immediately preceding sections of the course. In each case, the answer

to an SAQ appears at the end of the unit in which the SAQ occurs. If you get all the answers right, just move on to the next section. If you get any wrong, check back to see where you have gone wrong.

Select those phrases from list B which complete correctly the phrases given in A.

A

1 The CPU ...
2 The main characteristics of a computer system are ...
3 A machine code is ...
4 A machine code is an example of a ... language.
5 BASIC is an example of a ... language.
6 A BASIC interpreter ...
7 A computer program is ...

B

(a) low-level
(b) high-level
(c) ... holds data and instructions, controls its own processing, and controls the operation of input and output devices.
(d) ... a series of instructions or procedural steps for the solution of a specific problem.
(e) ... that it is capable of storing large quantities of data, is able to process this data very rapidly and lastly that it is able to store a program which controls its own operation.
(f) ... translates code written in BASIC into machine code.
(g) ... a code which corresponds directly with the electrical patterns within a computer.

1.5 Statement numbers

Let's have a closer look at Program 1 again:

```
10 INPUT FS
20 INPUT SC
30 LET SUM = FS + SC
40 PRINT SUM
50 END
```

Program 1 (from p11)

We have said that a program is a sequence of instructions. In the program above each line is an instruction. Thus:

```
10 INPUT FS
```

is the first instruction of the program, and

```
50 END
```

is the last. Instructions in a programming language are sometimes called statements. We will use the two words synonymously.

Entering statements

When sitting at the keyboard of your micro, the actual process of entering a statement is only carried out after the RETURN key has been pressed. So what happens is:

You type `10 INPUT FS` then press RETURN. Then you type `20 INPUT SC` and press RETURN etc.

You see on the screen

```
10 INPUT FS
20 INPUT SC
```

You will have noticed that each line begins with a number. These must be whole numbers in the range 1–9999, and they determine the order in which the instructions are processed (executed), i.e. they define the 'sequence' of the instructions. The execution of the instructions starts with the line of the lowest number and continues in the sense of increasing numbers until instructed otherwise, or until the end of the program is reached. (More about 'until instructed otherwise' and 'ending' later.)

Why then, you may ask, was the program not written as follows?

```
1 INPUT FS
2 INPUT SC
3 LET SUM = FS + SC
4 PRINT SUM
5 END
```

Program 2

Why not indeed! The program would have done the job perfectly well! However, as you will soon find out when writing programs, you need a certain amount of flexibility. In particular you need the opportunity to slip into the program a statement which you have overlooked, or one which will allow you to make an important modification. Numbering our lines 10, 20, 30, 40 and 50 leaves nine empty lines between statements which may be used to correct or modify the program. When running, the processing proceeds to the next highest line number of the program, so the gap of nine unused line numbers does not slow down the program execution in any way.

SAQ 2

Look at the line numbers of the following programs and decide which programs would produce the correct sum of FS ('first') and SC ('second').

(a)

```
11  INPUT FS
59  INPUT SC
93  LET SUM = FS + SC
401 PRINT SUM
500 END
```

Program 3

(b)

```
23 INPUT FS
32 INPUT SC
49 LET SUM = FS + SC
40 PRINT SUM
50 END
```

Program 4

(c)

```
10 INPUT FS
20 INPUT SC
15 LET SUM = FS + SC
40 PRINT SUM
50 END
```

Program 5

(d)
```
100 INPUT FS
200 INPUT SC
110 LET SUM = FS + SC
190 PRINT SUM
220 END
```
Program 6

(e)
```
100 INPUT FS
50 INPUT SC
407 LET SUM = FS
902 PRINT SUM
1000 END
```
Program 7

1.6 Executions and commands

The command RUN

Execution? No, it's not the end but the beginning! Let's get on and run our first program before we get tired of it!

So far we have entered Program 1:

```
10 INPUT FS
20 INPUT SC
30 LET SUM = FS + SC
40 PRINT SUM
50 END
```
Program 1 (from p11)

And what happened? Nothing. This is because the computer is waiting for us to give instructions to the program as a whole. If you want to execute this program, you must give it the command RUN. This you put on a new line as follows:

```
10 INPUT FS
20 INPUT SC
30 LET SUM = FS + SC
40 PRINT SUM
50 END
RUN
```
Program 1 (from p11)

(Don't worry about RUN not having a line number – we'll explain that shortly.)

Then press RETURN. You will then see ? on the screen which is the computer's way of asking for data. Give it your first number; then press RETURN; another ? appears because the computer needs your second number. Give it the second number and press RETURN. The answer should now appear. Here is our version of this run:

```
10 INPUT FS
20 INPUT SC
30 LET SUM = FS + SC
40 PRINT SUM
50 END
RUN
? 12157
? 7896
20053

Ready
```

Figure 5 A complete run of a program

What we are doing, therefore, is to distinguish between the entry of a program and its execution. Let's go back to the dialogue between you and the child playing computers. A very explicit infant may have said 'I am going to give you two numbers, I want you to write them down, add them together, and then tell me their sum'. At this point you know exactly what to do, but you haven't yet done anything. You've got the instructions, though; you've been programmed. The dialogue may proceed thus:

CHILD: 'Start'
YOU: 'Give me the first number'.
CHILD: '12157'
YOU: 'Give me the second number'.
CHILD: '7896'
YOU: '20053'

Now these instructions have been carried out (run). A program then is just a set of instructions for the computer. When the program is run or executed these instructions are carried out.

Other commands: LIST, CSAVE, CLOAD
RUN is not the only command which you can give to a program as a whole. You can also use LIST, CSAVE, CLOAD as follows:

LIST
You may spend quite a lot of time typing a program into your machine and during that process make several corrections. You may then wish to see a fair copy of the program as a whole on the screen. If you type the word LIST a complete copy of the program in line number order will appear on the screen.

The command LIST will display a complete copy of the program; it will move up the screen when the screen is full (i.e. 'scroll') at quite a speed, so it can be useful to isolate a smaller part of the text:

```
LIST 20-130
```

will list the program from line 20 to line 130; you may use any two line numbers in ascending order. If the number of lines requested is greater than a screenful, the listing will scroll upwards until the last mentioned line number ends up at the bottom of the screen.

```
LIST -150
```

will list the program, from the beginning, as far as line 150.

```
LIST 40-
```

will list the program from line 40 to the end.

```
LIST 80
```

will just list the line requested – in this case line 80.

The main reason for wishing to manipulate the text of your programs in these ways is to enable you to check it to see if any alterations are needed. This process is known as editing.

There is a further option when listing the program: touching the space bar on the keyboard will stop the listing scrolling up the screen. Touching any key will restart the scrolling.

If you wish to delete a complete line from your program, type in the line number and then touch RETURN. If you then LIST the program again, you will find that the line in question will have been completely deleted.

If you make an error whilst typing in a program line, before touching the RETURN key, touching the delete (DEL) key will rub out the last character to have been keyed in. As the Oric's keys have a repeat facility, keeping the DEL key depressed will cause characters to be rubbed out until the key is released for the current line. An alternative method of removing a complete line before it has been entered (by touching the RETURN key), is to hold down both CTRL and X at the same time. A backslash is placed at the end of the line to show that the line has been ignored and the cursor (black square) moves to the next line on the screen.

Touching CTRL and L will clear the screen, making it easier to read your corrected listing.

If you wish to amend a line totally, just type the line number and the new statement, then touch RETURN. If you then LIST your program, you will find the line in question has disappeared to be replaced by your new instruction.

CSAVE

When you have developed a program to a satisfactory stage you may wish to take a copy of it onto tape; the CSAVE command will do this for you. This is necessary because it is quite a laborious task to key in a long program into memory each time that you need it; the computer's memory loses all the information it contains when the power is switched off.

You will need a suitable cassette recorder – the Oric will work with most portable mono cassette recorders as it has special filter circuits to even out differences in signal strength.

Connect the 'DIN' socket on the recorder to the seven-pin 'DIN' socket at the rear of the Oric's case using the proper lead. If your lead has a 'remote' connection (usually a 2.5 mm jack plug), you may be able to let the computer control the operation of the recorder.

Any low noise blank cassette tape should be suitable. To avoid wasting time winding on to the place on the tape where the program is held, it is best to use short tapes. Even so, it is useful to have a counter on your recorder to help monitor the position of recordings on tapes.

To store a program on tape, it must already be in the computer's memory. Type LIST then touch RETURN to check that the program is there. The listing should end with the message

```
Ready
```

Place your blank cassette into the recorder, set the tape counter to zero, and wind the tape just past the coloured leader tape. Set the tone and volume controls to roughly two-thirds of their maximum values. Type in CSAVE "PROGNAME" Instead of PROGNAME inside the quotes you would put the name you had chosen for your program. The name can be up to 17 characters long and include full stops, hyphens, etc. Set the recorder to record and play and touch RETURN. The program will then be saved onto the magnetic tape as audio tones.

```
Saving PROGNAME
```

will be displayed at the top of the screen and Ready will appear when the saving is complete.

You can practise without actually using a tape until you feel confident. When you are ready try it for real. The Oric usually saves programs at high speed (2400 baud), and is reliable. If you wish to be extra sure of a good recording you can save at the slow speed (300 baud), by typing in CSAVE "PROGNAME",S.

CLOAD
When you wish to load a tape program back into the computer, the command

```
CLOAD ''PROGNAME''
```

(should your program be called PROGNAME!) will enable the computer to search the tape and, when it has found the program, to load it. If you want to load the first program you come across, or if you've forgotten to make a note of the exact name of the program

```
CLOAD '' ''
```

will load the first program the computer comes across on the tape.

The message

```
Searching....
```

will be shown at the top of the screen until the program has been found, when the message will change to

```
Loading PROGNAME
```

and Ready will appear when the program is safely loaded into the memory of the computer.

If you saved a program at the slow speed, it must also be loaded at the same speed, and the command must be

```
CLOAD ''PROGNAME'',S
```

Words like LIST, RUN, CSAVE and CLOAD, which allow us to handle the program as a single entity, are called commands are are provided by the BASIC interpreter. A command occupies a line on its own and generally does not have a line number, e.g. the word RUN after line 50 of Program 1 causes the program to start to execute and is equivalent to the child's command 'start' in the dialogue above. Commands will be discussed in greater detail in a later unit but the four we have already looked at will allow us to get by for a start.

Computer responses
After a command has been successfully carried out the interpreter informs the user of this fact by writing a message on the screen: for the Oric this is

```
Ready
```

Keywords
These are the BASIC words which go in the program to specify what action is to occur, e.g. INPUT, PRINT, LET.

1.7 Execution and data

You will have realised that we have written a program which will add any two numbers in a quite general way, for it is immaterial to the program what numbers we enter when we receive prompts ? on the screen. As the computer executes the program it must be able to request that we input actual numbers for its particular task, i.e. it must have the facility to demand specific data to do the job in hand. You need to be able to distinguish clearly between the program, as a set of more or less general instructions, and the data which are the actual numbers which must be

input when the program is executing, in order to solve a particular problem. You can, of course, run your program repeatedly with many pairs of numbers, as you will see later.

Another way of looking at these instructions is to visualise the situation as an umpire gathers the runners at the start of a race in order to give them certain instructions: 'Go down the right-hand side of the field to the furthest corner, over the stile and turn left down the lane ...' The umpire's instructions are analogous to a program. If the runners understand what he is saying then they know what to do; but they are still at the starting line. They haven't actually started. This is analogous to the program having been entered into the machine. Then the umpire says 'Go!' and the race starts. This is analogous to the computer starting to execute the program. Let's extend the analogy and consider the cross-country race as a novelty race. Imagine that the umpire did not give enough instructions for the runners to complete the course but he said something like 'When you get to the bottom of the lane you will find further instructions pinned to the oak tree ...' These instructions should be sufficient to guide the runners over the next part of the course, i.e. on to the next clue, and so on until the end of the race. These clues are analogous to giving the program more data during the course of its execution. This analogy may help you to see the important distinction between entering a program into the machine, executing the program, and then inputting data during the course of its execution.

SAQ 3

Below is a printout from a computer. It contains keywords, commands, responses from the system and items of data. It also contains sections that are concerned with entry, execution and listing. Identify as many of these items as possible as follows:

Mark keywords with	K
Mark commands with	C
Mark system responses with	R
Mark data items with	D

Bracket lines concerned with entry
Bracket lines concerned with execution
Bracket lines concerned with list

```
10 INPUT FS
20 INPUT SC
30 LET SUM = FS + SC
40 PRINT SUM
50 END

RUN
? -37
? -46
-83
Ready

LIST

10 INPUT FS
20 INPUT SC
30 LET SUM = FS + SC
40 PRINT SUM
50 END
```

```
RUN
?  12.83
?  48.95
 61.78

Ready

RUN
?  17.0009
? -29.3629
 -12.362
```

Program 1 (from p11)

1.8 INPUT, PRINT and LET

You have seen that a program is a sequence of statements, and we have given you an intuitive idea of how each statement works. You may also have noticed that each of the three types of statement used so far (INPUT, LET and PRINT) corresponds to one of the three main devices which comprise the computer system (input, central processor and output devices) We will now look at each statement in more detail.

INPUT

The word INPUT is a signal to the computer that during execution, an item of data must be entered at the input device. We saw this happen when we ran our first program: after the ? we entered 12157, pressed RETURN to complete the input procedure and then found ourselves confronted by another ? requesting the input of the next number. What, then, happened to 12157, the FS item of data? The answer is that it has been stored for later use in the program's execution in the storage location labelled 'FS'. The word 'FS' has two main functions in the program, (a) when written and later referred to by the programmer, it reminds him that at this point in the program the first item of data should be input, and (b) when written in the statement 10 INPUT FS the word 'FS' is the name or label of a location in the computer's memory. So 10 INPUT FS means enter a number at the input device and store it in the location labelled 'FS'.

As briefly mentioned in Example 1, we have had to adopt names such as FS and SC because of the way that the Oric's BASIC interpreter looks at words. 'FIRST' and 'SECOND' would have been more meaningful but 'SECOND' causes an error as it contains the word 'ON'. Equally, using 'ONE' and 'TWO' would not work for the same reason. 'ONE' contains 'ON' as well! If, at any stage in your Oric programming, an inexplicable syntax error is indicated in one of your program statements, it would be wise to check, amongst other possibilities, if you have chosen store names which include reserved words from the BASIC language.

PRINT

The statement 40 PRINT SUM has almost the reverse effect to statements 10 and 20, in that it allows us to output information from the machine. It is a signal to the machine to take a copy of the contents of the store location labelled 'SUM' and pass it to the output device, which for most users of this course will be a television screen. Notice that LPRINT will literally result in a printed output if a printer is attached to your microcomputer but you still use the command PRINT when your microcomputer is attached to a television set for its output device.

LET

30 LET SUM = FS + SC is an example of an assignment statement. It is in this

type of statement that the processing takes place. As you can see, it is a mixture of store names ('SUM', 'FS', 'SC') and arithmetic operators (= and +). If you read it forwards, it says

> Let the store location 'SUM' be made equal to the contents of the location 'FS' added to the contents of the location named 'SC'.

However, like many mathematical expressions, it is often clearer when read from right to left of the '=' as follows

> Add the contents of the location 'FS' to the contents of the location 'SC' and store the result in the location labelled 'SUM'.

Generally the assignment statement has the form:

```
LET store location name = expression.
```

This means find the value of the expression of the right-hand side of the '=' and store this value in the store location named on the left-hand side of the '='.

The tricky point about LET ...=... is that it is easily confused with ...=... in mathematical equations. An example will demonstrate the difference. Suppose you have stored a number in location L and you want to make the number in that store 5 greater than it now is. You write:

```
LET L = L+5
```

Now obviously this doesn't mean:

```
L = L+5
```

since there is no value for L which could make this true. What it does mean is that the computer has added 5 to the number that is in store location L.

In Oric BASIC the word LET is optional in assignment statements.

LET SUM = FS + SC can be abbreviated to SUM = FS + SC

both statements meaning that the computer is to add together the numbers in store FS and in store SC and to place the answer in store SUM.

Later on in the course we have left out the word LET in assignment statements to cut down on the typing for long programs. LET is used in this the earliest part of the course.

1.9 Store locations

As we have already said, one of the main characteristics of a computer system is its capacity for storing large quantities of data. We must now consider how the BASIC language allows us to allocate store location names. If you look at our first program and recall that a computer is capable of only doing one thing at a time, it is fairly obvious that when we reach line 20 and wish to input our second number, the number that we entered in line 10 must have been stored somewhere! In this case the first number was stored in the location labelled 'FS'. We can think of the storage locations as being like a set of pigeon-holes where we distinguish clearly between the label or address or name on each pigeon-hole and the contents of the hole.

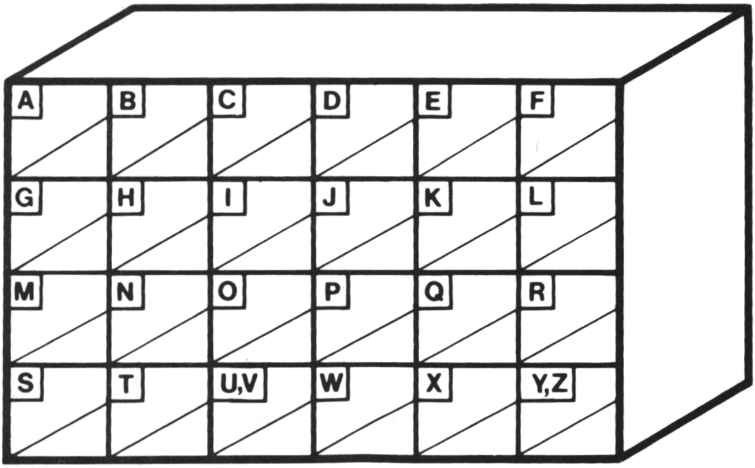


Figure 6 A model of the store locations in a very small computer

You will see that in our model of possible store locations we have used the labels A, B, C ... On the other hand, in the program we have been studying we have used words of up to three characters to label our stores, e.g. 'FS', 'SC'. This brings us to our first major point of difference between the interpreters for various microcomputers now available. Some BASIC interpreters allow a much wider range of storage names than others. Some machines limit your variable name to a few characters. Others allow longer variable names or even names of unrestricted length. The Oric will accept variable names of greater than two characters, but only pays attention to the first two, so that HIT and HILL will be treated as the same store location – HI.

Choosing store location names

Clearly it makes life a lot easier for the programmer if he chooses store location names which remind him of what he is storing. That is why we chose FS, SC and SUM. We could have used A, B and S so that our program would have been:

```
10 INPUT A
20 INPUT B
30 LET S=A+B
40 PRINT S
50 END
```

Program 8

[K] When you see this symbol it means we suggest you try this program on your own microcomputer, if you have one. To do this, key in the lines, touching RETURN after each, finally type RUN and then press RETURN. Your computer will ask you for a number. Give it one and touch RETURN. It then asks you for the second number. Give it the second number, touch RETURN and the sum will appear on your screen. If we had done this, then after inputting 12157 and 7896, the store locations would be:

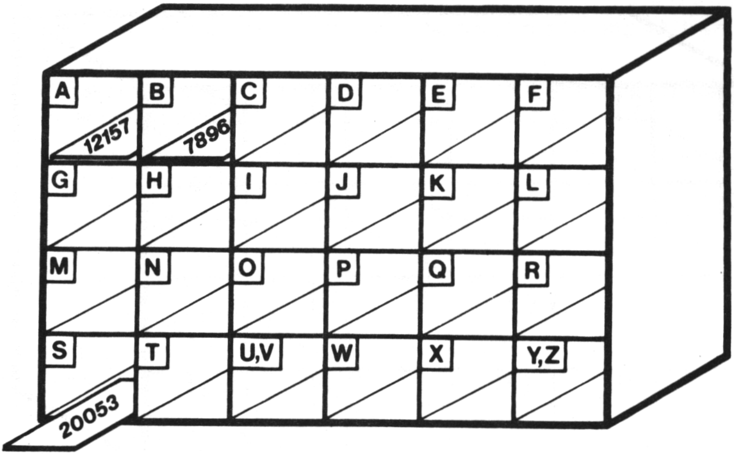


Figure 7 State of the store locations after Program 8

The common system of location names
Since long store names are not available on many microcomputers, we will, from now on, use the system of location names that works on practically every microcomputer, until we come to lists in a later unit. The system we shall use labels a location by a letter of the alphabet followed by a digit. This gives us 286 possible locations, as shown in Figure 8.

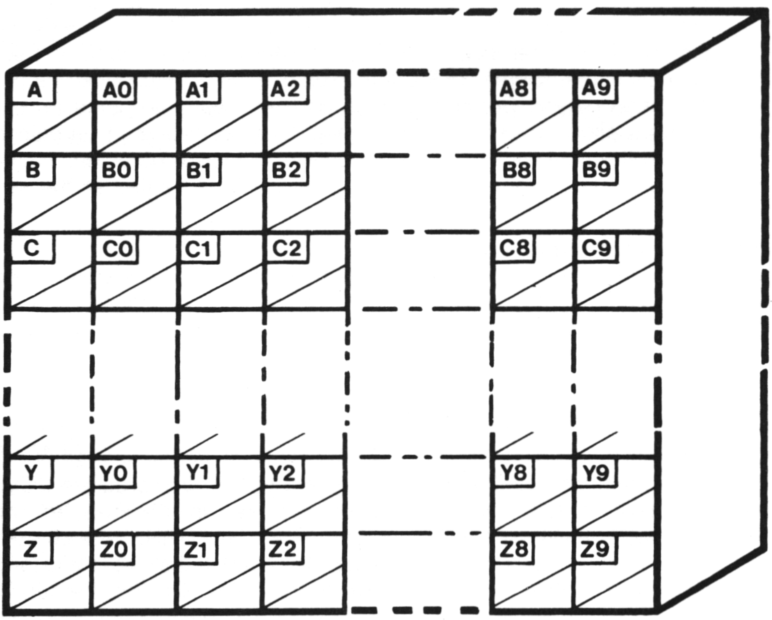


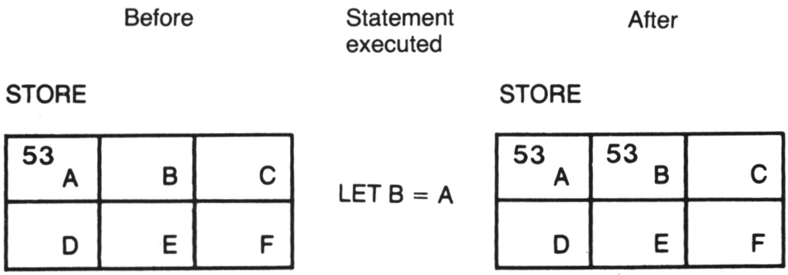
Figure 8 286 possible store locations

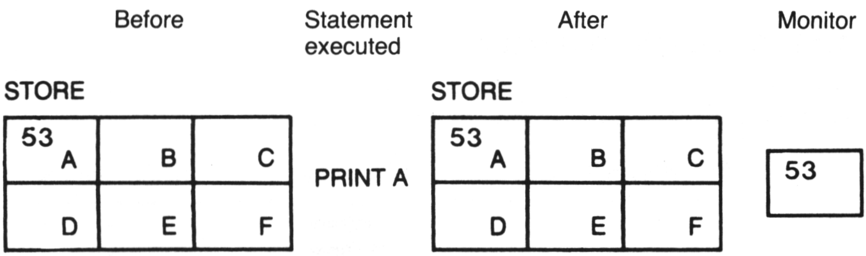
1.10 Copying and overwriting

BASIC statements can have two different effects on the contents of a store location. Or rather a statement can either have no effect on the contents of the location or it can change the contents. This is illustrated below.

Effect of copying

Suppose we have the number 53 stored in location A. What happens after `LET B=A` and after `PRINT A`? In each case A still stores the number 53 after the statement has been executed:

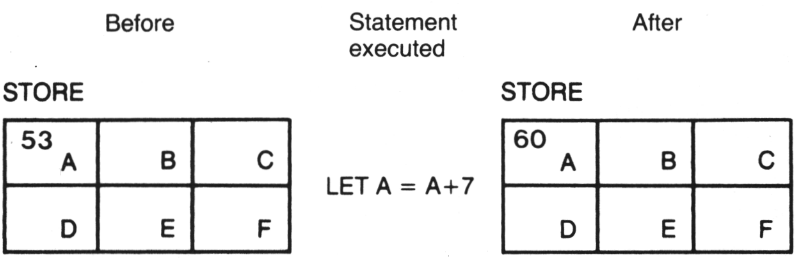




In each case the copying statements leave the original store location unchanged. It's just like getting a statement of your bank account: the piece of paper copies your account but your account still has your money in it!

Effect of overwriting

Suppose now that we still have the number 53 stored in location A but this time execute the statement `LET A = A+7`. The result is:



The statement `LET A = A+7` overwrites the contents of A. That is, the original contents disappears and is replaced by the new contents, which is, in this case, 60. Of course we could have made the new contents of A to be 60 in many ways, e.g. by, say: `LET A = 60`.

SAQ 4
Which of the following are valid store location names for numbers, according to the rules given on pages 21-23

(a) N3
(b) 3N
(c) W10
(d) B#
(e) QJ
(f) M
(g) M5
(h) M–5
(i) M+5
(j) UO

Give reasons for discarding those names which you reject.

1.11 Arithmetic operators

When you do arithmetic you use four main operators: +, –, × and ÷. BASIC has the same operators, although two are printed differently:

Everyday symbol	Meaning	BASIC symbol
+	add	+
–	subtract	–
×	multiply	*
÷	divide	/

SAQ 5
Write the following expressions using BASIC symbols for the arithmetic operators. (Where the expressions use brackets, leave the brackets in your answers.)

(a) 3+7
(b) 3×7
(c) 8÷4
(d) 5×(2+8)
(e) 30÷(3+2)
(f) 24–(4×3)
(g) 5×6×7
(h) 81–(27×2)

SAQ 6
If A has the value of 2, B has the value of 5 and C has the value of 10, calculate the values of the following:

(a) A+B+C
(b) A*B
(c) A*B*C
(d) C/A
(e) C/(B-A)
(f) A*A
(g) (B*C)/(B-A)
(h) (C-B)*(C+B)

The arithmetic is actually done (executed) in BASIC through assignment

statements (LET statements) which tell the computer's arithmetic unit (part of the central processor) what to do. We can illustrate this with the following computer model.

Effect of LET A = B – C

LET A = B – C

STORE AT START

A	15 B	10 C
D	E	F

Remember to read this from the right to the left of the '=' sign. It says take the number in location B, subtract from it the number in location C and put the result in location A. So the result is:

Result of LET A = B – C

Notice that the contents of B and C are unchanged.

STORE AT FINISH

5 A	15 B	10 C
D	E	F

Effect of LET A = B * C

STORE AT START

A	15 B	10 C
D	E	F

STORE AT FINISH

150 A	15 B	10 C
D	E	F

SAQ 7

Fill in the values in the store locations A, B and C after each line has been executed in these programs.

1\. Program — Store location values

Program	A	B	C
`10 LET A = 12`	☐	☐	☐
`20 LET B = 5`	☐	☐	☐
`30 LET C = A*(A + B)`	☐	☐	☐
`40 LET A = A + 10`	☐	☐	☐

2. Program — Store location values

Program	A	B	C
`10 LET A = 20`			
`20 LET B = A * 3`			
`30 LET C = A / 4`			
`40 LET A = B + C`			

What you have just done isn't (we hope) difficult and it doesn't get any more difficult when we move on to more complicated store location names. We have to make the names more complicated because A, B, C ... only gives us 26 stores and, as we said on page 21, we are going to use location names A, A0, A1, etc. So,

```
LET P4 = Q1 * R1
```

is no different from `LET A = B * C`. P4, Q1 and R1 are simply store location names. P4 is one name, just as XYZ 823A is one car number.

We are now ready to use the arithmetic capacity of a computer.

Example 2
Write a BASIC program to enter two numbers into the computer and then output their sum, difference, product and quotient.

Solution
This may look complicated but we've really solved this already. We had a program (Program 1) to output the sum of two numbers, so to output their difference, product and quotient, we only need to change the arithmetic operator in line 30 which reads

```
30 LET SUM = FS + SC
```

First, however, we will rewrite the program using the shorter store location names:

Original version

```
10 INPUT FS
20 INPUT SC
30 LET SUM = FS + SC
40 PRINT SUM
50 END
```

New version

```
10 INPUT N1
20 INPUT N2
30 LET S = N1 + N2
40 PRINT S
50 END
```

Program 9

Now what we need is three extra versions of the new version, each with a different line 30:

```
10 INPUT N1
20 INPUT N2
30 LET S = N1+N2
40 PRINT S
50 END
```

```
10 INPUT N1
20 INPUT N2
30 LET D = N1 - N2
40 PRINT D
50 END
```

```
10 INPUT N1
20 INPUT N2
30 LET P =N1*N2
40 PRINT P
50 END
```

```
10 INPUT N1
20 INPUT N2
30 LET Q = N1/N2
40 PRINT Q
50 END
```

(Using D for the location for difference, P for the location for product, and Q for the location for quotient.)

Do we need to write four programs? Fortunately no, because when we copy the numbers from locations *N1* and *N2*, we don't destroy these contents so we can use them four times over in one program:

```
 10 INPUT N1
 20 INPUT N2                ——— Original program for SUM
 30 LET S = N1 + N2
 40 PRINT S
 50 LET D = N1 - N2         ——— Extra lines for difference
 60 PRINT D
 70 LET P = N1 * N2         ——— Extra lines for product
 80 PRINT P
 90 LET Q = N1 / N2         ——— Extra lines for quotient
100 PRINT Q
110 END
```

Program 10 Sum, difference, product and quotient of two numbers

$\boxed{K}$Key Program 11 into your microcomputer. Then key `RUN` and input two numbers. A typical print out should look like:

```
RUN
? 57.82
? 19.11
76.93
38.71
1104.9402
3.025641

Ready
```

1.12 Numerical constants

Earlier in this unit we saw that our first BASIC program was capable of manipulating whole, decimal and negative numbers. At this stage we won't go into detail on how numbers are represented in BASIC, but just show you that we can use numbers directly in assignment statements.

The statement `LET P = 427*R` means create the number 427, multiply it by the number found in store location R and then store the result in location P.
(Don't be put off by thinking that computers only handle binary numbers. The computer's interpreter enables us to input ordinary decimal numbers.)

Similarly, the statement `LET Y4 = 3.142+Z8` creates the number 3.142 and adds it to the contents of location Z8, and then stores the sum in location Y4.

And the statement `LET A = -48.93/B` creates the number −48.93 and divides it by the number found in location B and then stores the result of this calculation in location A.

Exercise preamble
Progress in metrication has been slow and life still abounds with irritating little conversions which occasionally tease us, e.g. pounds weight to kilogrammes, yards to metres, pints to litres, a knitting pattern with balls of wool in ounces which must be bought in grams, etc. ... If we go on holiday we mentally convert kilometres

into miles, and pounds sterling into other currency. Most of us still think of body and weather temperatures in terms of degrees Fahrenheit rather than Centigrade or Celsius. We can imagine the home microcomputer of the future having a general conversion program in it which will do all these diverse conversions for us. The next two exercises are on writing programs to do conversions. You only need the ideas introduced in the earlier programs in this unit.

Exercise 1
Write programs in BASIC to carry out each of the following conversions:

(a) Input a number representing a length in inches, and output this length in centimetres, given that one inch is equivalent to 2.54 centimetres.

(b) Input a number representing a weight in ounces, and output that weight in grams, given that one ounce is equivalent to 28.375 grams.

(Answers to exercises appear at the end of each unit with the SAQ answers.)

Exercise 2
Any conversion involves 'conversion factor × number to be converted' so it is possible to write a general conversion program where you input two numbers each time you use it: the conversion factor and the number to be converted. Write a general conversion program which will do this.

1.13 The remark statement: REM

The statement `REM` is the remark statement. It allows us to give a title to a program or make some other meaningful remark about the program. For example, within the body of a program it helps us identify what the program or section of the program does. The `REM` statement is not executed by the computer and is there purely for the benefit of either the programmer or user, i.e. when the computer sees `REM` at the beginning of a line it ignores everything on that line.

The next program is concerned with calculating percentages and so as a title to the program our first statement will be `10 REM **Percentage Calculation**`. The `**` have no function other than to emphasise the title. We also use `REM` to explain pieces of the program which may be difficult to understand, as you will see later.

Example 3
Write a BASIC program to input two numbers and output the second as a percentage of the first.

Reminder Percentage = (second ÷ first)×100.

Solution

```
10 REM ** PERCENTAGE CALCULATION **
20 INPUT F
30 INPUT S
40 LET P = (S/F) * 100
50 PRINT P
60 END
```

(Program title using REM statement)

Program 11 Percentage calculation

Typical runs

```
RUN
? 57
? 74
129.82456

Ready

RUN
? 74
? 57
77.027027

Ready
```

[K] Program 11.

1.14 More complicated arithmetic

We have reached the stage when we can use the computer like a simple four-function calculating machine, but we will soon wish to do slightly more complicated arithmetic. Generally BASIC allows us to set out equations in a familiar way. We can use brackets, i.e. ( ) to group together certain values, and when BASIC evaluates an expression it deals with the values inside the brackets first. Next come values involving multiplication or division and finally, addition and subtraction.

This order of preference for performing arithmetic operations is discussed more fully in a later unit, but you will soon see that this order just formalises the way we naturally go about arithmetic calculations.

Let's show you what we mean.

Example 4
Write the following expressions in BASIC:

1. AB+C 2. A(B+C) 3. $\frac{A}{B+C}$

Solutions

1. A*B+C
   The order of precedence rules tell us that A * B will be evaluated first and the C added. If you were worried about this you could write (A * B) + C but the brackets aren't essential here.
2. A*(B+C)
   Notice that, just as a bracket is needed in A(B+C) so it is needed in A*(B+C).
3. A /(B+C)

Now try some for yourself.

SAQ 8
Write the following as BASIC expressions:

1. ABC 2. $\frac{AB}{C}$ 3. $\frac{A+B}{C}$

Exercise 3
Now that you have written the expressions in SAQ 8 as BASIC expressions, write a program that will allow you to input three numbers (A, B and C) and print out the values of the expressions in SAQ 8.

1.15 Literal printing

You have seen already that we can print out the values from store locations. You will find as the course progresses that the PRINT function is very versatile. One use of this statement is to print messages on the monitor screen, which will be helpful to the user when the program is running. These messages are usually referred to as prompts. We have seen already that when an input statement is encountered during the execution of a program, a `?` appears on the screen to remind us that an input is required. In even slightly complicated programs, a series of `?` s on the screen is confusing to the user since he may not know which input value the `?` is prompting. Prompts generated by PRINT statements are very useful in these circumstances.

It is very easy to get a computer to print a reminder or message on the screen. All you need is a line such as:

```
20 PRINT "Message"
```

This simply prints

```
Message
```

on your screen.

In other words, whatever appears between the quotes thus " " after the word PRINT will be printed out exactly as it stands. Notice that, as in the case of the `REM` statement, the computer doesn't execute the words within the quotes. Thus

```
20 PRINT "A + B"
```

results in

```
A + B
```

on your screen and the computer does not add the value in location A to the value in location B .

The following example demonstrates the use of PRINT " " to remind the programmer and user of what the program is doing.

Example 5
Write a BASIC program to convert a temperature value given in degrees C into degrees F.

Remember °F = (9 / 5) * °C + 32
where F = temperature in Fahrenheit
and C = temperature in Centigrade.

Solution

```
10 REM ** CENTIGRADE TO FAHRENHEIT**
20 PRINT "Enter next temp in degrees C"
30 INPUT C
40 LET F = (9/5) * C + 32
50 PRINT "This temp in degrees F is"
60 PRINT F
70 END
```

Print message preceeds input statement so that the message is printed before ? appears

Program 13 *Temperature conversion*

Typical run

```
RUN
Enter next temp in degrees C
? 16
This temp in degrees F is
60.8

Ready
```

[K] Program 12.

Lower case letters are obtained by touching CTRL and T; touching CTRL and T a second time will return you to the 'all capitals' mode – note that variable names and BASIC keywords must be in capital letters.

Assignment 1

NEC students: your solution to this assignment should be sent to your NEC tutor for marking. If you own your own microcomputer, write your own assignment in the BASIC for that computer and tell your tutor what make of microcomputer it is. *FlexiStudy students*: complete the assignments according to the instructions given to you by your FlexiStudy centre.

Remember to make good use of remark and literal print statements when writing your programs.

1. If you deposit £D in an account paying P% rate of interest for one year, then the yield at the end of the first year is given by the equation

$$Y = D \times \frac{P}{100}$$

(a) Write a BASIC program to input values for D and P and to output the yield, Y.
(b) If the original deposit together with the accrued interest is left in the account for a further year at the same rate of interest then the compound interest after the second year will be given by the equation

$$C = (D + Y) \times \frac{P}{100}$$

Extend your program for (a) to calculate and output this compound interest.

2. Consider the problem of estimating the cost of installing replacement aluminium double glazed windows. The windows comprise three parts:

(a) a hardwood surround, (b) aluminium frame and (c) glass. If the height of the window is H metres and the width is W metres, then the total lengths of both hardwood and aluminium required are given approximately by the expression (2H + 2W) metres; and the area of glass required by the expression (H × W) square metres.

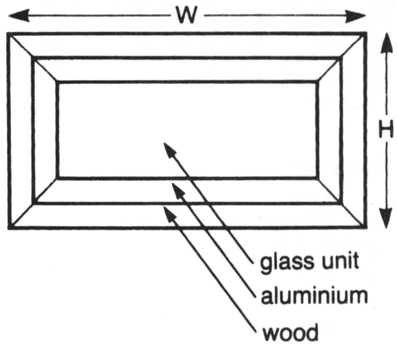


Figure 9

Write three separate BASIC programs which, on being given values for height and width in metres, will output the cost of

(a) the hardwood surround if the wood costs £3 per metre;
(b) the aluminium surround if the aluminium costs £4 per metre;
(c) the glass unit if the glass unit costs £40 per square metre.

Now link these three into one program to estimate and output the total cost of installation, if the labour cost is £50 per window.

Objectives of Unit 1

Now that you have completed this unit, check that you are able to:

Write simple programs using:
- Line numbers ☐
- INPUT ☐
- LET ☐
- PRINT ☐
- Store locations identified by a single letter or a letter followed by a single digit ☐
- Copying from one location to another ☐
- Overwriting ☐
- +,–,*,/ ☐
- ( ) ☐
- Numerical constants ☐
- REM ☐
- PRINT "   " ☐

Know when to use:
- RETURN ☐
- RUN ☐

Know how to respond to:
- Ready ☐
- error message ☐
- ? ☐

Answers to SAQs and Exercises

SAQ 1

A	B
1	(c)
2	(e)
3	(g)
4	(a)
5	(b)
6	(f)
7	(d)

SAQ 2

(a) and (e) would run as Program 1.
(b) is asked to print SUM before SUM has been calculated.
(c) and (d) are asked to calculate SUM before SC has been input.

SAQ 3

```
                                      K
       5 REM **SAQ 3**                          ]
      10 INPUT FS                               ]
      20 INPUT SC                               ]  Entry
      30 LET SUM = FS + SC                      ]
      40 PRINT SUM                              ]
      50 END                                    ]
  C— RUN                                        ]
  R— ? -37 ----------------------------- D      ]
  R— ? -46 ----------------------------- D      ]  Execution
       -83 ----------------------------- R      ]

  R— Ready

  C— LIST
                                      K
       5 REM **SAQ 3**                          ]
      10 INPUT FS                               ]
      20 INPUT SC                               ]  Listing
      30 LET SUM = FS + SC                      ]
      40 PRINT SUM                              ]
      50 END                                    ]

  R— Ready

  C— RUN                                        ]
  R— ? 12.83 --------------------------- D      ]
  R— ? 48.95 --------------------------- D      ]  Execution
       61.78 --------------------------- R      ]

  R— Ready

  C— RUN                                        ]
  R— ? 17.0009 ------------------------- D      ]
  R— ? -29.2629 ------------------------ D      ]  Execution
       -12.362 ------------------------- R      ]

  R— Ready
```

(Have you noticed that this program has coped with negative and decimal fractional numbers?)

SAQ 4

(a) OK
(b) No, begins with a digit instead of a letter.
(c) Not allowed on all systems. (10 is two digits, not 1 as in, say, W8.)
(d) No, # is not an acceptable symbol in a variable name.
(e) No, uses two letters. (This will, of course, work on some machines.)
(f) OK
(g) OK
(h) No, '–' is not an acceptable symbol.
(i) No, '+' is not an acceptable symbol.
(j) OK

SAQ 5

(a) 3 + 7 (b) 3 * 7 (c) 8 / 4 (d) 5 * (2 + 8) (e) 30 / (3 + 2)
(f) 24 – (4 * 3) (g) 5 * 6*7 (h) 81 – (27 * 2)

SAQ 6

(a) 17 (b) 10 (c) 100 (d) 5 (e) 10/3 or 31/3 or 3.33 ...
(f) 4 (g) 50/3 or 16⅔ or 16.66 ... (h) 75

SAQ 7

1.

A	B	C
12		
12	5	
12	5	204
12	5	204

2.

A	B	C
20		
20	60	
20	60	5
65	60	5

Exercise 1

(a) Program 13

```
 5 REM ** PROGRAM 13**
10 INPUT L1
20 LET L2 = 2.54 * L1
30 PRINT L2
40 END
```

Typical runs

```
RUN
?  12        first use
30.48

Ready

RUN
?  36        second use
91.44

Ready
```

Ⓚ Program 13.

(b) Program 14

```
 5 REM ** PROGRAM 14**
10 INPUT W1
20 LET W2 = W1 * 28.375
30 PRINT W2
40 END
```

Typical runs

```
RUN
?  10          first use
283.75

Ready

RUN

? 50           second use

1418.75

Ready

RUN
?  16          third use
454
```

🄺 Program 14

Exercise 2

Program 15

```
 5 REM ** PROGRAM 15 **
10 INPUT V
20 INPUT F
30 LET N = F * V
40 PRINT N
50 END
```

Typical runs

```
RUN
?  16          Use for ounces
?  28.375      to grams
454

Ready

RUN
?  36          Use for inches to
?  2.54        centimetres
91.44
```

🄺 Program 15.

SAQ 8

1. A * B * C
2. A * B / C (A * B) / C is also correct, although the brackets aren't necessary.
3. (A + B) / C

Exercise 3

Program 16

```
  5 REM ** PROGRAM 16 **
 10 INPUT A
 20 INPUT B
 30 INPUT C
 40 LET R = A * B * C        ]- Calculates first expression and prints it out
 50 PRINT R
 60 LET R = (A * B) / C      ]- Calculates second expression and prints it out
 70 PRINT R
 80 LET R = (A + B) / C      ]- Calculates third expression and prints it out
 90 PRINT R
100 END
```

Notice that we can use 'R' as the location for all three answers because we copy (print out) each answer in turn before we overwrite with the next answer.

Typical runs

```
RUN
?  13
?  -27
?  55.2
-19375.2
-6.35869565
-.253623188

Ready

RUN
?  13
?  13
?  13
2197
13
2

Ready
```

```
 RUN
?  13
?  13
?  13
 2197
 13
 2
```

🄺 Program 16

UNIT 2
Making decisions

2.1 Introduction

The programs which we considered in Unit 1 were quite straightforward. They started their processing at the statement with the lowest number and continued in line number order until execution finished at the line with the highest line number. One thing that computers are very good at is lots of repetitive calculations; another is their ability to make decisions. Both of these features involve changing the sequence in which a program is executed. This unit will introduce you to some of the statements which enable you to write programs of this type. But first we are going to introduce you to a new type of PRINT statement.

2.2 PRINT ...,

In Unit 1 we wrote a program (Example 3) to output one number as a percentage of another. On the screen, the calculation and result appeared in the format:

```
RUN
? 57
? 74
129.824561
```

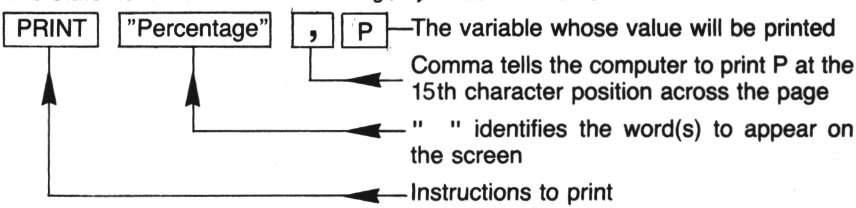
Obviously it would be better if the answer included the word 'Percentage' so that it was clear what was happening. This can easily be done by changing line 50 from 50 PRINT P to

```
50 PRINT "Percentage",P
```

The effect of this is:

Version of line 50	**Result on screen**
`50 PRINT P`	`129.824561`
`50 PRINT "Percentage",P`	`Percentage    129.824561`

The statement `PRINT "Percentage",P` has four items in it.



We can use PRINT ..., to improve the percentage program from Unit 1. At the same time we can improve the appearance of the program by making use of the literal print statement PRINT " " which we introduced in section 1.15.

Original program

```
10 REM ** PERCENTAGE CALCULATION **
20 INPUT F
30 PRINT F
40 INPUT S
50 PRINT S
60 LET P = (S / F) * 100
70 PRINT P
80 END
```

New program

```
10 REM ** PERCENTAGE CALCULATION **
15 PRINT "What is the first number"
20 INPUT F
25 PRINT "What is the second number"
30 INPUT S
40 LET P = (S/F) * 100
50 PRINT "Percentage",P
60 END
```

Program 1 Improved percentage program

Original program: typical run

```
RUN
?  80
?  37
46.25
```

New program: typical run

```
RUN
What is the first number  <———— effect of line 15
?  80
What is the second number <———— effect of line 25
?  37
Percentage   46.25        <———— effect of line 50
```

three spaces beyond the end of the previous item

Notice how the use of literal print at lines 15 and 25, together with PRINT " ", makes the program much more friendly and understandable when in use.

[K] Program 1.

Commas in PRINT statements

You can use commas in print statements to space out your results on the screen. BASIC usually has five PRINT zones across the screen:

72 characters

Zone 1	Zone 2	Zone 3	Zone 4	Zone 5
0 to 14	15 to 29	30 to 44	45 to 59	60 to 71

The Oric is different, however. It is supposed to put a space five characters long between items separated by a comma (and hence ten characters when two spaces are used, etc.) but in practice it only appears to leave a gap of three characters, or four between numeric fields. When LPRINT is used instead of PRINT, to output to the line printer instead of the screen, the first comma gives one character and all subsequent commas add an extra five characters' gap!

Thus

```
PRINT "Zone 1","Zone 2", "Zone 3"
```

results in

```
Zone 1   Zone 2    Zone 3
```

and

`PRINT "Percentage",P` gave `Percentage   46.25`

whereas

`PRINT "PERCENTAGE";P` would give `Percentage46.25`

The semi-colon (;) is not essential in Oric BASIC, but helps improve the readibility

of complex statements, and is required in some versions of BASIC so we shall generally include it.

SAQ 1
What would be the result on the screen from these print lines?
(Assume A=48, B=8, C=6 in these questions).

(a) `PRINT "Area";A`
(b) `PRINT "Length";B,"Width";C,"Area";A`
(c) `PRINT "Length","Width","Area"`

Because of the peculiarity of the tabulation used by the Oric, it wil normally be necessary to use the comma in the PRINT statement to space out the data fields in a table, but to use spaces in a literal print statement, for the column titles, to ensure that the titles and data columns line up neatly. This is unfortunate and hopefully future versions of Oric BASIC will overcome this problem.

2.3 Repetitions and GOTO

Suppose now that you wanted to use Program 1 to calculate all the percentages for a test taken by a whole class of pupils. You would have to use the program over and over again, starting a RUN each time, e.g.

```
RUN
What is the first number
? 80
What is the second number
? 42
Percentage    52.5

Ready

RUN
What is the first number
? 80
What is the second number
? 19
Percentage    23.75

Ready
```

Figure 1 Repeated use of percentage program

It would be much easier if, after the computer has calculated the first percentage, it went back to the beginning of its calculation and asked us for the next mark. This we can make it do using the statement

```
GOTO line number
```

which redirects the program to whichever line number we insert. Here is the percentage program rewritten in this way:

```
 10 REM ** MARKS INTO PERCENTAGES **
 20 PRINT "Input the total possible marks"
 30 INPUT T
 40 REM ** START OF INPUT **
 50 PRINT "Input the next mark"
 60 INPUT M
 70 LET P = (M/T) * 100
 80 PRINT "Percentage",P
 90 GOTO 40
100 END
```

Program 2 Percentage program for repeated use

Notice the new lines 20 and 30 which ensure that we only have to enter the maximum mark on the test once. The calculation is carried out in lines 60 to 80 and, in reading line 90, the program returns to line 40 to ask us for another mark. Note that we have branched back to a REM statement, which a) tells us which part of the program we have now jumped to; b) means that we can alter subsequent lines of coding without having to worry if we have messed up any GOTO addresses. The importance of this will become clear in longer and more complex programs 41.

A typical run is:

```
RUN
Input the total possible marks
? 80
Input the next mark
? 42
Percentage     52.5
Input the next mark
? 67
Percentage     83.75
Input the next mark
? 19
Percentage     23.75
Input the next mark
? 55
Percentage     68.75
Input the next mark
?
```

If you continue to input marks, the screen will fill, the display will scroll upwards removing earlier displayed information.

The `GOTO` statement interrupts the program's normal execution in line number order. As soon as the program reads `GOTO` it unconditionally transfers control to the line number in the statement. It is sometimes called an 'unconditional jump'.

[K] Program 2. (Touch CTRL and C when you are fed up, to break into the program run and stop the program).

SAQ 2

The following program squares numbers (i.e. multiplies a number by itself). Add a `GOTO` line to allow you to use the program over and over again to square successive numbers.

```
 5 REM ** CALCULATING SQUARES **
10 INPUT N
20 LET S = N * N
30 PRINT S
40 END
```

Program 3 Calculating squares

[K] Key and run lines 5 to 40. Add your extra line and run your new program.

2.4 Programming style

The use of the GOTO statement helped in some ways. However, it still left some loose ends, such as the ? at the end of the run. On reaching line 90 in Program 2 control is always returned to line 40 which then generates the demand for a further input; hence the ?. The program is then locked in a perpetual loop from which it cannot escape. The only way to stop this program is to break out of it which is accomplished by pressing CTRL and C.

To have the execution of the program left as it were in mid-air is obviously bad style, but we will sort this out in a little while. More importantly, we ought to warn you of the dangers of using GOTO. As we have said, this statement allows you to jump or branch to virtually any position in the program, which may at this stage seem to be a useful facility. But because GOTO allows us to jump rather at random to any point in the program, it is often used in such a way that the logical structure of the solution is broken up by 'jumps of convenience' to other parts of the program rather than by following the logical stucture of the analysis of the problem. We will, therefore, use the GOTO statement sparingly throughout this course. We shall only use it when we think the clarity of the program will be marred if it is not used. We hope that you will also try to follow our example and use the GOTO statement as little as possible. Though we will avoid its indiscriminate use, you will find it more widely used in some text books and computer magazines.

A reminder:
You will notice that we will nearly always make the GOTO line refer back to a REM line. This will allow us to put an explanation of the jump in the REM statement, to make the jump in program logic more easy to understand when looking at the program listing. Even more importantly, this will make amendments to program listings simpler; we will be able to add or remove lines, often without changing the details of the GOTO statement.

2.5 IF ... THEN ...

The problem that we have just left is how to signal to the computer that we have reached the end of the list of marks. When doing a manual calculation, we can see that we have reached the end, or, if not, we would have carried out some sort of counting procedure. In a little while we will see how the computer may be used to count for us, but first we will introduce a means of signalling that the end of the list has been reached.

One method is to end the list of numbers that you put in with a special number that will 'stick out like a sore thumb', e.g. −9999. We would hardly expect a pupil to have obtained −9999 marks in any test! This value is called a dummy or terminating value, or sometimes a rogue value. We want the program to run as normal when 'proper' marks are inputted but to stop when the mark −9999 is inputted. In other words, we want to be able to write a program with the following logical structure:

1. Start.
2. Input the total marks.
3. Input the next mark.
4. If this value is equal to −9999 then go to line 8 otherwise carry on to line 5.
5. Calculate the percentage.
6. Output the percentage.
7. Go to line 3.
8. Stop.

Figure 2 Stopping the percentage calculation

Fortunately there is a BASIC statement which will carry out the decision in line 4. It is:

```
IF ... THEN line number
```

Condition to be satisfied for program to jump to given line number

So all we need to do is to translate statement 4 in Figure 2 as

```
90 IF M = -9999 THEN 150
```

and make it part of Program 4. This statement means: if the value found in *M* is equal to −9999, then go to line 150, otherwise continue executing the next statement after 90. The statement in Figure 2 'otherwise carry on to line 5' is not translated into BASIC but it is implied: either jump out of sequence or carry on in sequence. We can do this very conveniently by using some new line numbers between the ones we have been using, when necessary. This gives us:

```
 10 REM ** PERCENTAGES **
 20 REM ** TOTAL MARKS **
 25 CLS
 30 PRINT "What is the highest mark"
 40 INPUT T
 50 REM ** START OF 'NEXT MARK' LOOP **
 60 PRINT "What is the next mark"
 70 INPUT M
 80 REM ** -9999 MEANS 'ALL FINISHED' HERE **
 90 IF M = -9999 THEN 150
100 REM ** CALCULATION **
110 LET P = (M/T) * 100
120 PRINT "Percentage",P
130 REM ** GO BACK FOR NEXT INPUT **
140 GOTO 50
150 REM **ALL DONE **
160 END
```

Program 4 Percentage calculation with a terminating value

You use Program 4 in exactly the same way as Program 2 until you have put in the last mark to be converted to a percentage. Then you enter the mark −9999 and the program ends.

Note the effect of line 25:

```
25 CLS
```

CLS stands for 'Clear the screen'. It does exactly that – only your program output then appears on the cleared screen.

Here is a typical run.

```
RUN
What is the highest mark
? 80
What is the next mark
? 43
Percentage     53.75
What is the next mark
? 29
Percentage     36.25
What is the next mark
? 62
Percentage     77.5
What is the next mark
? -9999 <------------------------terminating mark

Ready
```

[K] Program 4 and use it to convert some marks of your own. Terminate your run with −9999.

SAQ 3
You ended SAQ 2 with the program:

```
 5 REM ** SAQ 3 **
10 INPUT N
20 LET S = N * N
30 PRINT S
40 GOTO 5
50 END
```

but like the percentage program, this never stops. Modify the program to include a dummy value for terminating the program.

2.6 Inequalities

It was helpful to be able to use the expression `M = -9999` in the `IF ... THEN ...` statement to determine whether or not the branch should occur. The statement means if `M = -9999` is true then branch, otherwise carry on. The '=' states a relationship between `M` and `-9999`.

BASIC allows expressions to include relationships:

Relationship	**Example**	**Meaning**
$>$	$A > B$	The value in store location A is greater than the value in B
$<$	$X < Y$	The value in store location X is less than the value in Y

True or false?
Consider the expression $A < B$. If $A = 2$ and $B = 5$ then $A < B$ is true, because 2 is less than 5. Consider the same expression with values $A = 2$ and $B = 1$. Now $A <$

B is false, because 2 is not less than 1. Similarly if $A = 2$ and $B = 2$ then $A < B$ is false, for 2 is not less than 2. In writing programs we often find it useful to be able to know whether a statement involving $=$ or $<$ or $>$ will be true or false. This is called the logical state of the assertion, e.g.

Assertion	**Logical state**
$3 > 2$	True
$7 < 7$	False

You will probably find this easy enough for positive whole numbers but may be less sure of what happens in other cases. If in doubt, remember the number line:

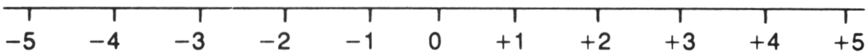


Figure 3 The number line

If a number is found on this line to be to the left of a second number, then the first is less than the second number; if to the right then the first is greater than the second.

Example 1
Test whether the following expressions are true or false for the given values of A and B.

Values A	Values B	Expression
2	5	A>B
2	−5	A>B
−2	−5	A>B
−2	−1	A>B
−5	2	A<B
5	−2	A<B
−3	3	A=B

Solution
To do this we work out the value of each expression using the values given and then use the number line to decide whether or not the assertion is true or false for those particular values. So the solution is:

Values		Assertion		
A	B	Expression	Its value	Its logical state
2	5	A>B	2> 5	F
2	−5	A>B	2>−5	T
−2	−5	A>B	−2>−5	T
−2	−1	A>B	−2>−1	F
−5	2	A<B	−5< 2	T
5	−2	A<B	5<−2	F
−3	3	A=B	−3= 3	F

F = false T = true

SAQ 4
Complete the following table to determine whether the given expressions are true or false for the values given.

Values		Assertion		
A	B	Expression	Its value	Its logical state
3	7	A>B		
5	3	A>B		
−3	5	A>B		
8	5	A<B		
3	9	A<B		
8	−2	A<B		

We are now in a position to use relationships to allow control of a program to jump to a new line when certain conditions are satisfied.

Example 2
In the following program segment, after executing line 30, will control pass to line 40 or line 100?

```
10 LET A=-3
20 LET B=2
30 IF A + B > 0 THEN GOTO 100
40
```

Program 5

Solution
$-3 + 2 > 0$ is false, so the branch to 100 will not occur and control will just pass on to line 40.

SAQ 5
In the following program segments, after executing line 30, will control pass to line 40 or to line 100?

(a)
```
10 LET A = 7
20 LET B = -8
30 IF A-B<0 THEN GOTO 100
40
```

(b)
```
10 LET X = 3
20 LET Y = -3
30 IF X/Y = -1 THEN GOTO 100
40
```

(c)
```
10 LET P = -1
20 LET Q = -3
30 IF P/Q = -1 THEN GOTO 100
40
```

(d)
```
10 LET M = 3
15 LET N = -4
20 LET P = -2
30 IF M-N<N-P THEN GOTO 100
40
```

(e)
```
10 LET R = 1
20 LET S = -2
30 IF R+S>-1 THEN GOTO 100
40
```

Programs 6–10

2.7 Flowcharts

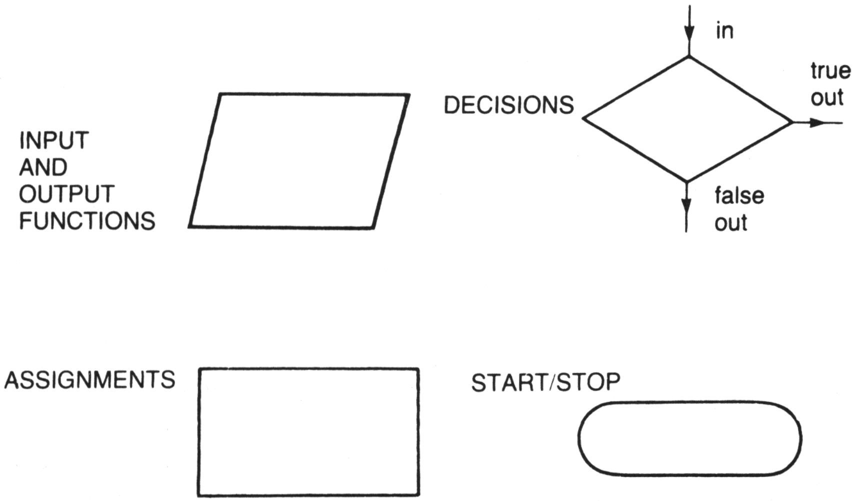
As we have said, the principal task for a programmer is to find a suitable way of expressing the solution strategy to solve a particular problem. At this stage we must introduce you to what must be the ugliest word in computer jargon: algorithm. This word is used to mean a general solution strategy, and is defined as a series of instructions or procedural steps for the solution of a specific problem. You will notice that in this case the computer is not mentioned. Apart from that, the definition of program and algorithm are identical. A program, then, is an algorithm written for a computer.

There are three basic ways of stating an algorithm:

(i) a description
(ii) BASIC coding
(iii) a flowchart

Flowcharts are a bit like blueprints and appeal to those of us who like to see events displayed in pictorial, chart or cartoon form.

We display the different functions within an algorithm by using different shaped boxes.



An → shows the sequence of the algorithm, and the boxes contain appropriate scripts.

The first program from Unit 1 can be expressed in flowchart form as follows:

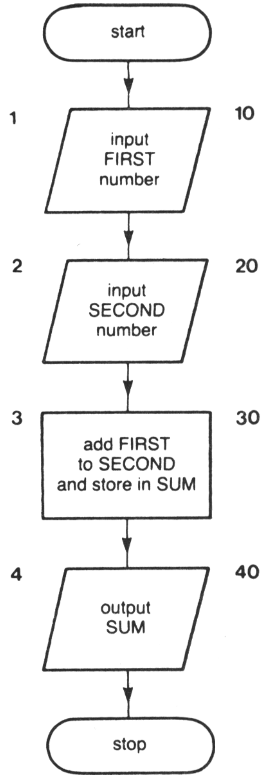


Figure 4 Flowchart of Program 1 from Unit 1

The descriptions of the functions in the boxes are in cryptic-English but could be followed by someone who has no knowledge of BASIC. In that respect we say that we try to keep these descriptions language independent. The numbers on the left-hand side of the boxes refer to the statements in the descriptive algorithm in Figure 4 of Unit 1, and the numbers on the right-hand side of the boxes refer to the statements in the BASIC program in Program 1 of Unit 1.

SAQ 6
Construct a flowchart for the percentage program (Program 1).

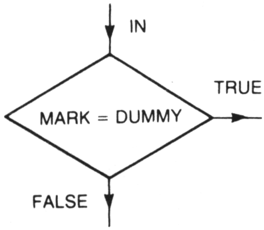
The decision box
You have seen how decisions are effected in BASIC using the `IF ... THEN ...` statement. The logic is:

IF assertion is true THEN go to line X
otherwise (assertion is false) carry on to the next line

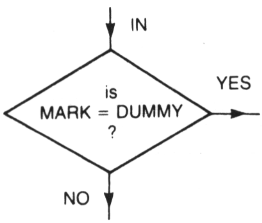
The basic idea of branching to line X or carrying on to the next line is depicted in the flowchart by two exit lines from a decision box. The decision of line 90 of Program 4:

```
90 IF M = -9999 THEN 150
```

could be depicted in language independent form



The assertion MARK = DUMMY may be expressed as a question



Flowchart style is up to you. The test of the effectiveness of a flowchart is whether or not you can follow the flowchart easily some time after its composition! Or, if you are trying to communicate your ideas to somebody else, whether they can follow your flowchart easily.

A flowchart for Program 4 is given below:

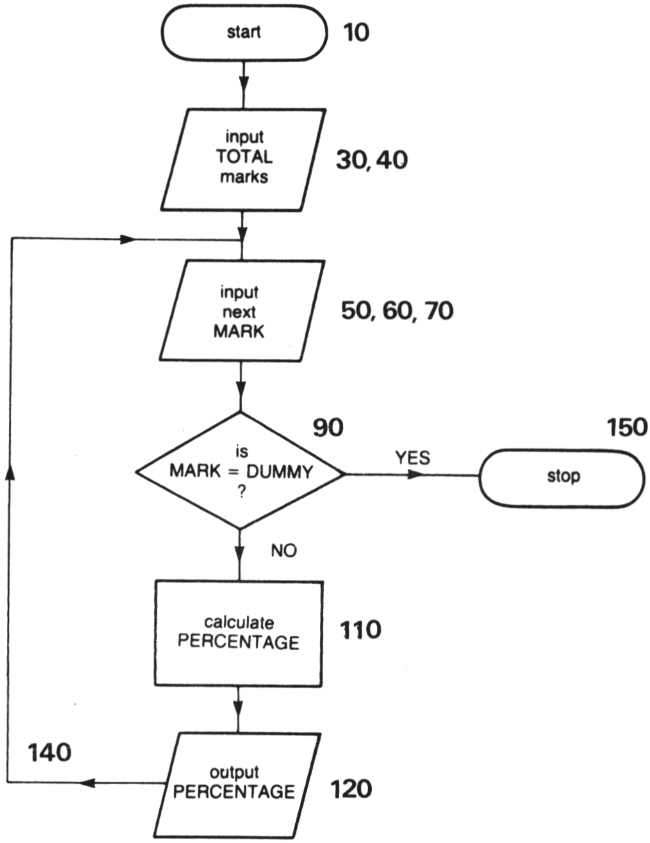


Figure 5 Flowchart for percentage program

The numbers on the right-hand side of the flowchart boxes refer to the statement numbers in the program. `Statement 140 GOTO 50` is represented by a loop back to box 50.

SAQ 7

Write a flowchart for the program you wrote in answer to SAQ 3.

Now that we have introduced the idea of flowcharts, we can use them to help plan the sructures of the programs that we are going to write.

2.8 Counting

As we have said, computers are good at carrying out lots of repetitive procedures. If, however, we wish to control these activities rather than just start and stop them, as we did in the last example, then we must use the computer to count the repetitions for us. If we carry out a specified number of repetitions of an activity, we start counting at the first activity, add one for each subsequent activity, until we reach a predetermined limit. We can depect this procedure in flowchart form.

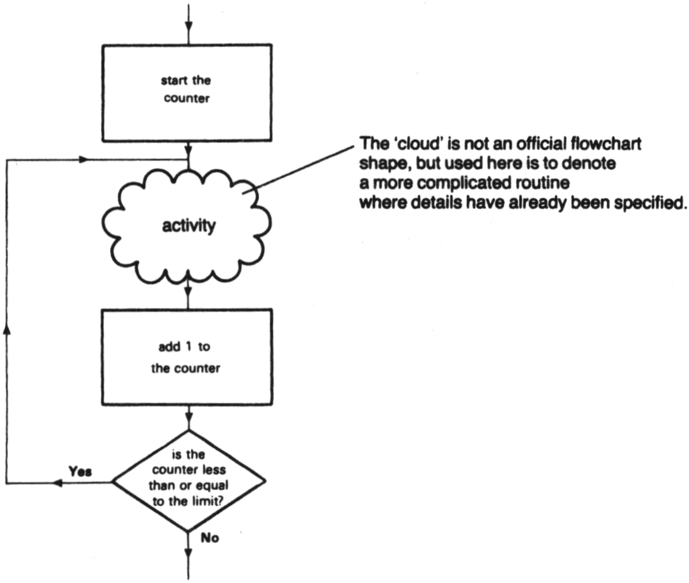


Figure 6 A counter in a flowchart

Notice that there are three parts to the counter:

(i) the procedure that sets the counter to its initial value;
(ii) the procedure for adding 1 to the counter each time the activity is completed;
(iii) the procedure for stopping the counter and leaving the activity when it has been executed the required number of times.

Great care must be taken to ensure that we exit from such a repetitive loop at exactly the point we wish to. As a warning, note that though this loop has counted up to 10, the value of the location COUNT on leaving the loop will be 11. This is a point which you would have to be very careful about if you wished to use the number in COUNT later on in the program.

SAQ 8
How many numbers will be input with the flowcharts?

(a)

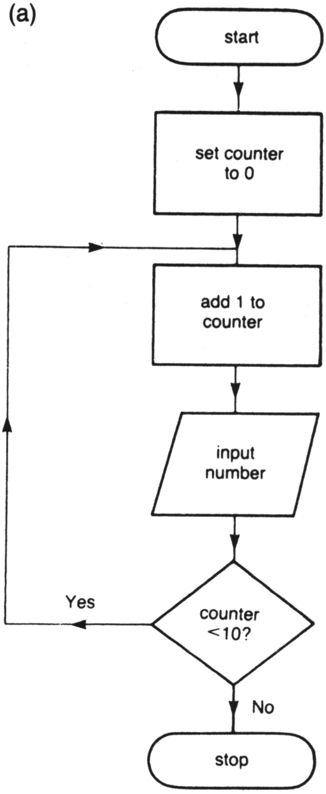


Figure 7a

(b)

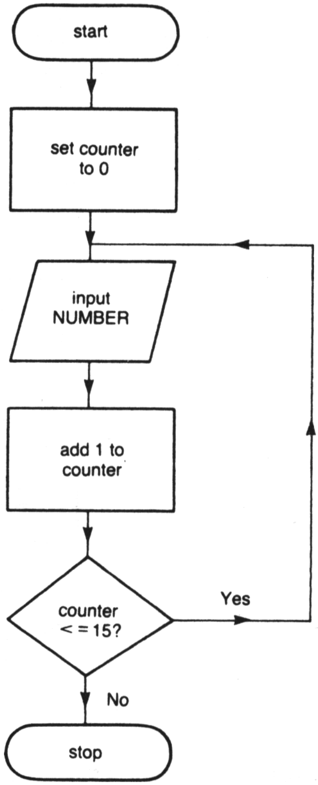


Figure 7b

SAQ 9
Make the following program read five numbers by completing the `IF ... THEN ...` statement.

```
 5 REM **SAQ 9**
10 LET C = 0
20 INPUT N
30 IF C=   THEN 60
40 LET C = C + 1
50 GOTO 20
60 END
```

Program 11

Example 3
Write a BASIC program to calculate and output the percentage marks for a group of five pupils.

Solution
If we assume that the 'activity' in the cloud in the flowchart of Figure 6 was

input the mark calculate the percentage output the percentage

then we can display the algorithm from this solution in flowchart form.

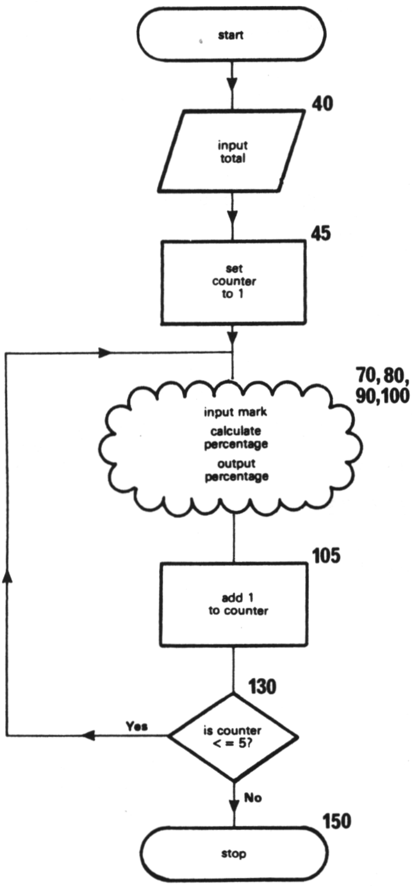


Figure 8 Flowchart for percentage calculation on five marks

This tells us the structure of the program that we need to write. The actual program can now be written by modifying Program 4 to incorporate the counter. So the required program is:

```
 10 REM ** PERCENTAGES **
 20 REM ** Total marks **
 25 CLS
 30 PRINT "What is  the highest mark"
 40 INPUT T
 45 C = 1
 50 REM ** START OF 'NEXT MARK' LOOP **
 60 PRINT "What is the next mark"
 70 INPUT M
 80 REM ** CALCULATION **
 90 P = (M / T) * 100
100 PRINT "Percentage",P
105 C = C + 1
110 PRINT "line 110"
120 PRINT "count here =",C
130 IF C < = 5 THEN 50
140 REM ** ALL DONE **
150 END
```

Start the counter: 'initialisation'

Add 1 to the counter: 'incrementation'

We've added this line to check what is happening to the counter at this point in the program.

Counting complete?

Program 12 Adding a counter to the percentage program

```
RUN
What is the highest mark
? 75
What is the next mark
? 57
Percentage    76
line 110
count here =  2
What is the next mark
? 62
Percentage    82.666667
line 110
count here =  3
What is the next mark
? 43
Percentage    57.333333
line 110
count here =  4
What is the next mark
? 39
Percentage    52
line 110
count here =  5
What is the next mark
? 70
Percentage    93.333333
line 110
count here =  6
```

Note: On emergence from the loop the value of the counter is 6

$\boxed{K}$ Program 12.

Exercise 1

In question 2 of Assignment 1, you wrote a program to calculate the cost of double glazing a window. If you wanted to use the program to calculate the costs of

several windows, you would have to run the program again and again. This exercise asks you to modify the program to cover more than one window.

Notice that the 'activity' for repetition will be:

input height and width of window
calculate cost of installation of this window
output the cost

(a) Draw a flowchart algorithm to calculate and output the cost of each of six windows.
(b) Code the algorithm in BASIC. [K] your answer and try a run for six windows on your microcomputer.
(c) Extend your program to cope with any chosen number of windows, to be specified at the beginning of the program.

Exercise 2
Extend the problem posed in Question 1(b) of Assignment 1. The 'activity' for repetition will be:

calculate the yield
output the year and its yield
calculate the deposit for the next year

(a) Draw a flowchart algorithm to calculate and output the yield for each year up to six years. Check your answer.
(b) Code this algorithm in BASIC in full detail. Check your answer. [K].
(c) Extend the algorithm to calculate and output the yield for each year up to any chosen number of years, to be specified at the beginning of the program. Check your answer. [K].

2.9 Comparisons

We have seen how the BASIC language allows us to compare two numbers. We often wish to decide whether a particular value is larger or smaller than another. This is a process which is fundamental to sorting items of data. Throughout the course we will consider sorting methods in some detail, so let's start with the simplest case.

Example 4
Devise an algorithm in descriptive form to input two numbers and to output the larger of the two.

Comment
We have expressed our algorithms as flowcharts throughout most of this unit so this time we will use the descriptive method introduced in Unit 1.

Solution
1. Start.
2. Input first number.
3. Input second number.
4. If first number > second number then go to 7 otherwise carry on to 5.
5. Output second number.
6. Go to 8.
7. Output first number.
8. Stop.

This is not the neatest solution, but is close to the layman's 'first attempt' at the problem. We will seek neater solutions when we return to sorting methods in a later unit.

Assignment 2

1. Devise a flowchart and write a BASIC program to input two numbers and output the smaller of the two. Modify the program so that it will process (a) five pairs of numbers, (b) any number of pairs of numbers.

2. Extend the 'mark – percentage' algorithm on page 43, and express it in the form of flowchart and BASIC program:

(a) to accommodate a class of any size;
(b) to calculate the average percentage mark;
(c) to pick out the highest mark.

Objectives of Unit 2

Now that you have completed this unit, check that you are able to:

Combine literal printing and variable print in `PRINT` statements □

Use `,` to space `PRINT` statements □

Use `GOTO` to repeat the use of a program □

Use a dummy value to terminate a program □

Use `IF ... THEN ...` □

Find the logical state of assertions including >, <, = □

Construct flowcharts □

Insert counters in flowcharts and programs to control the repeated use of part of a program or flowchart □

Answers to SAQs and Exercises

SAQ 1

(a) `Area48`
(b) `Length8   Width6   Area48`
(c) `Length   Width   Area`

SAQ 2

All you need to do is add

```
35 GOTO 5
```

SAQ 3

```
 5 REM ** SAQ 3 **
10 INPUT N
15 IF N = -9999 THEN 45
20 LET S = N * N
30 PRINT S
40 GOTO 5
45 REM ** ALL DONE **
50 END
```

Program 13

(Note that this terminating value is not quite as satisfactory as using −9999 as a dummy mark. You can't have a mark of −9999, but you might perhaps want to square −9999; this program would refuse to do it.)

SAQ 4

Values		Assertion		
A	B	Expressions	Its value	Its logical state
3	7	A>B	3>7	F
5	3	A>B	5>3	T
−3	5	A>B	−3>5	F
8	5	A<B	8<5	F
3	9	A<B	3<9	T
8	−2	A<B	8<−2	F

If you got any of these wrong, look at them again on the number line

A ______________ B **A to the left of B means A<B true**

B ______________ A **A to the right of B means A>B true**

SAQ 5

(a) 40 (b) 100 (c) 40 (d) 40 (e) 40

Your computer could help solve this problem for you. The statements

```
    40 PRINT "40"
```

and

```
   100 PRINT "100"
```

will cause the appropriate line to be output. The following programs show how you could have solved (a) and (b) above.

Program to solve (a)

```
  5 REM ** PROGRAM 14 **
 10 LET A = 7
 20 LET B = -8
 30 IF A - B < 0 THEN 100
 40 PRINT "40"
 50 GO TO 999
100 PRINT "100"
999 END
```

Program 14

Program to solve (b)

```
  5 REM ** PROGRAM 15 **
 10 LET X = 3
 20 LET Y = -3
 30 IF X / Y = -1 THEN 100
 40 PRINT "40"
 50 GOTO 999
100 PRINT "100"
999 END
```

Program 15

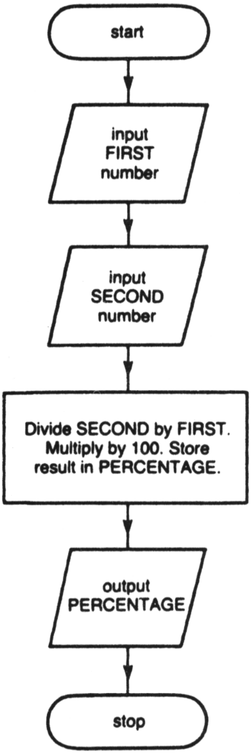
Effect of running Program 14

```
RUN

40
Ready
```

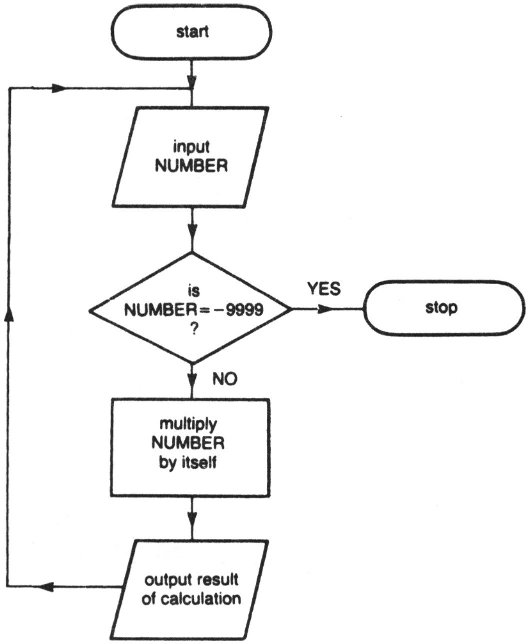
Effect of running Program 15

```
RUN

100
Ready
```

SAQ 6



SAQ 7



SAQ 8
(a) 10 (b) 16

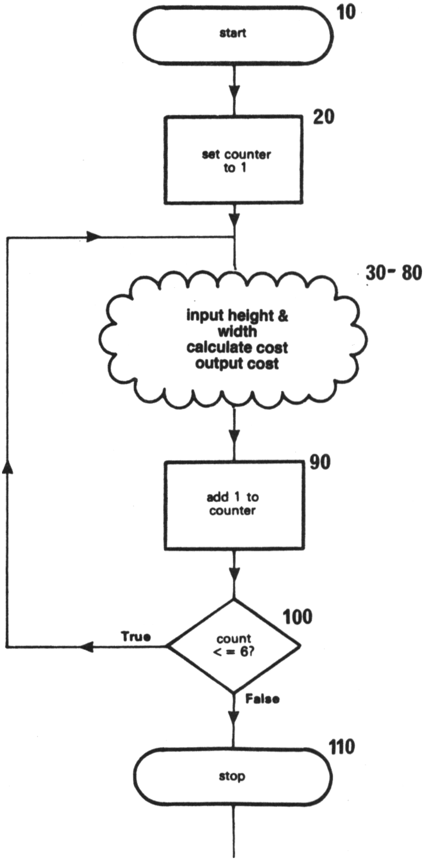
SAQ 9
30 IF C = 4 THEN 55

Full listing:

```
 5 REM **SAQ 9**
10 LET C = 0
15 REM ** READ NEXT NUMBER **
20 INPUT N
30 IF C = 4 THEN 55
40 LET C = C + 1
50 GOTO 15
55 REM ** ALL DONE **
60 END
```

Exercise 1

1(a)



1(b)

```
 10 REM **Cost of double glazing**
 20 LET C = 1
 25 REM ** NEXT INPUT **
 30 PRINT "Enter height (in metres)"
 40 INPUT H
 50 PRINT "Enter width (in metres)"
 60 INPUT W
 70 LET K = 14 * (H + W) + 40 * H * W + 50
 80 PRINT "Window ";C;" cost ";K
 90 LET C = C + 1
100 IF C <= 6 THEN 25
110 END
```

Program 16

Lines 20, 90 and 100 are the counter

```
RUN
Enter height in metres
? 1.5
Enter width in metres
? 2
Window 1 cost 219
```

```
Enter height in metres
? 1.5
Enter width in metres
? 3
Window 2 cost 293
Enter height in metres
? 1.5
Enter width in metres
? 4
Window 3 cost 367
Enter height in metres
? 2.5
Enter width in metres
? 2
Window 4 cost 313
Enter height in metres
? 1
Enter width in metres
? 2
Window 5 cost 172
Enter height in metres
? .5
Enter width in metres
? 1
Window 6 cost 91
Ready
```

1(c)

```
 10 REM ** COST OF DOUBLE GLAZING **
 12 PRINT "Enter number of windows ";
 14 INPUT N
 16 PRINT
 20 LET C = 1
 25 REM ** NEXT INPUT **
 30 PRINT "Enter height (in metres)";
 40 INPUT H
 50 PRINT "Enter width (in metres)";
 60 INPUT W
 70 LET K = 14 * (H + W) + 40 * H * W + 50
 80 PRINT "Window ";C;" cost " ;K
 85 PRINT
 90 LET C = C + 1
100 IF C <= N THEN 25
110 END
```

Program 17

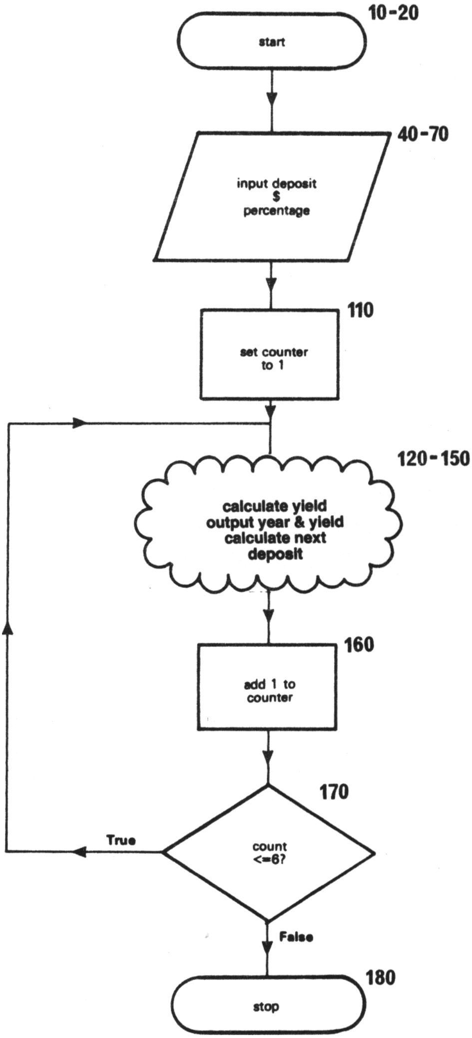
If the keyword PRINT is entered as a program statement, without anything else after it, the result is that a blank line is printed on the screen. This serves to space the lines of output in your display, and is useful because solid blocks of words are both unattractive and more difficult to read.

```
RUN
Enter number of windows 3
Enter height in metres
? 1
Enter width in metres
? 1.5
Window 1 cost 145

Enter height in metres
? 2
Enter width in metres
? 1
Window 2 cost 172
```

```
Enter height in metres
? 2.5
Enter width in metres
? 1
Window 3 cost 199

Ready
```

Exercise 2

2(a)



2(b) Program

```
 10 REM ** COMPOUND INTEREST **
 15 CLS
 20 PRINT "Compound Interest Table"
 30 PRINT
 40 PRINT "How much will you deposit £";
 50 INPUT D
 60 PRINT "What percentage interest (%)";
 70 INPUT P
 80 PRINT
100 REM  ** CALCULATE/PRINT TABLE **
110 C = 1
120 REM ** START OF CALCULATION LOOP **
130 Y = (P * D) / 100
140 PRINT "Year ";C;" --- Yield --- £";Y
150 LET D = D + Y
160 LET C = C + 1
170 IF C <= 6 THEN 120
180 END
```

Separates table from above input lines

Program 18

```
RUN
Compound Interest Table

How much will you deposit £
? 100
What percentage interest (%)
? 15%

Year 1 --- Yield --- £15
Year 2 --- Yield --- £17.25
Year 3 --- Yield --- £19.8375
Year 4 --- Yield --- £22.813125
Year 5 --- Yield --- £26.2350938
Year 6 --- Yield --- £30.1703578
```

Ready

Note the use of the £ sign and the % sign, and the way the table is laid out. It is little things like this which add to the 'user-friendliness' of a program and its output.

2(c) Program

```
 10 REM ** COMPOUND INTEREST **
 15 CLS
 20 PRINT "Compound Interest Table"
 26 PRINT
 28 PRINT "How many years";
 30 INPUT N
 40 PRINT "How much will you deposit (£)";
 50 INPUT D
 60 PRINT "What percentage interest (%)";
 70 INPUT P
 80 PRINT
100 REM ** CALCULATE/PRINT TABLE **
110 C = 1
120 REM ** START OF CALCULATION LOOP **
130 Y = (P * D) / 100
140 PRINT "Year ";C" --- Yield --- £";Y
150 D = D + Y
160 C = C + 1
170 IF C <= N THEN 120
180 END
```

Program 19

```
RUN

Compound Interest Table

How many years? 4
How much will you deposit (£)
? 1000
What percentage interest (%)
? 13.75

Year 1 --- Yield --- £137.5
Year 2 --- Yield --- £156.40625
Year 3 --- Yield --- £177.912109
Year 4 --- Yield --- £202.375024

Ready
```

UNIT 3
Strings

3.1 What is a string?

The first two units were concerned with processing numbers. The layman often sees the computer as a 'number cruncher' but this is certainly not the main function of a computer, especially in a commercial environment. In this unit we will see how a computer may be used to manipulate characters, using the BASIC language.

By character we mean the alphabet in lower case (small letters) and upper case (capitals), the ten digits 0-9, the space character punctuation marks and some special characters, as follows:

!	@	#	$	%	^	&	*	(	)	#	+	:
1	2	3	4	5	6	7	8	9	0	-	=	\
Q	W	E	R	T	Y	U	I	O	P	[	]	
q	w	e	r	t	y	u	i	o	p	(	)	
A	S	D	F	G	H	J	K	L	:	"		
a	s	d	f	g	h	j	k	l	;	'		
Z	X	C	V	B	N	M	<	>	?			
z	x	c	v	b	n	m	.	.	/			

You have learnt to write programs using numbers (3, 57, -92, etc.) and variables (A, G, Z, etc.). BASIC also allows us to enter characters into the computer in groups.

These groups of characters are referred to as strings. Some examples of strings are:

`CAT`	(a word)
`Margaret Thatcher`	(a name)
`z9)?27`	(a mixture of characters)
`ABS 123 W`	(a car registration number)

i.e. a string can be any mixture of characters – even a space is a very important character in a string!

As far as BASIC is concerned, a number is treated as a number when it is to be used to do some arithmetic, otherwise it is considered to be a string of numeric characters. When we look at a car or telephone number we see it as a group of numeric characters; we would not use this collection of digits to do any serious arithmetic.

Store locations for strings

How then can we signal to the computer that the group of characters which we are entering should be treated as numbers for arithmetic purposes, or just a string of characters? The distinction is made in BASIC by how we label the storage locations into which we put the characters. If a store location name is followed by the symbol $ then the characters which are entered into that location are treated as strings of characters.

You saw in Unit 1 that Oric BASIC accepts names for store locations for numbers as follows:

The store location names may be:

(a) a single letter (e.g. A, B, etc.)
(b) two letters (e.g. AB or XY, etc.)
(c) a single letter followed by a single number (e.g. A1, B3 etc.)

If you have a store location name which is longer than two characters, only the first two characters are counted; subsequent characters are ignored. For example, the

store location name NUMBER will be treated as if it were NU; A12 will be treated as if it were A1, and so on.

The same rule applies for locations in store which hold string variables; a dollar sign ($) is added after the variable name, to signify a string.

The store location names may be:

(a) a single letter, followed by $; (e.g. A$, B$, etc.)
(b) two letters, followed by $; (e.g. AB$, XY$, etc.)
(c) a single letter, followed by a single number, followed by $; (e.g. A1$, B3$, etc.)

Again, if you have a store location name which is longer than two characters plus $, the first two characters are counted; subsequent characters before the $ are ignored. For example, the store location name WORDS$ will be treated as if it were WO$; B76$ will be treated as if it were B7$, and so on.

Thus you can now think of a microcomputer as having two areas for store locations: one for numbers and one for characters. This is illustrated in Figure 1.

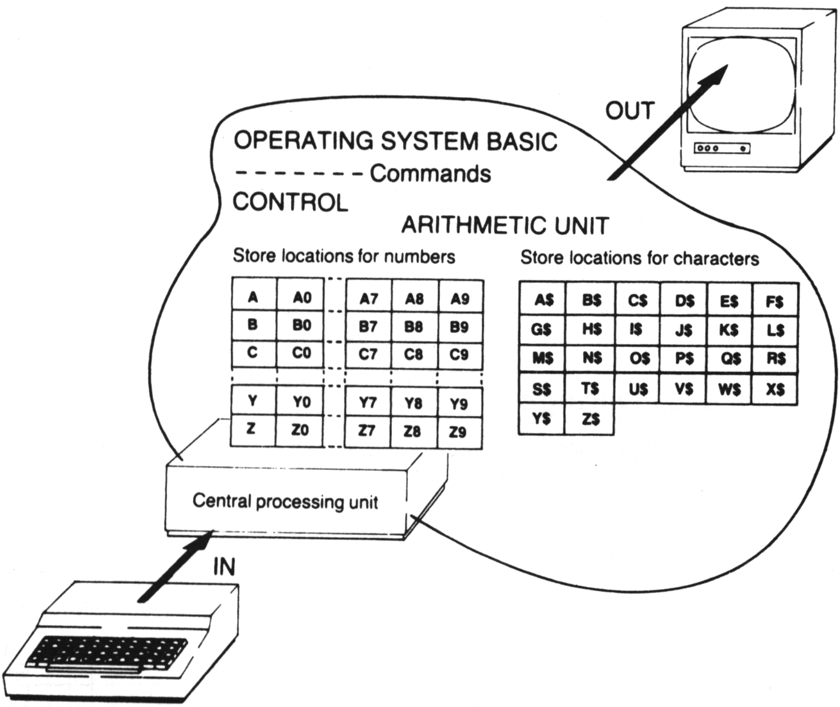


Figure 1 A summary of our system so far

" " with strings

Usually we have to show the computer that we want our string of characters to be treated as a string. To do this we put "   " around the string. Thus we write

```
10 LET Q$="Hello"
```

and

```
30 IF Q$="Hello" THEN 80
```

and so on.

SAQ 1

Which of the following are valid store location names for strings:

(a) A$ (b) M8 (c) T7$ (d) B9 (e) C$3 (f) 8P$ (g) 2$ (h) 5$L

SAQ 2

Which of the following are correct BASIC statements:

(a) `LET A = 87`
(b) `LET B$ = "Fred"`
(c) `LET M$ = 9583`
(d) `LET K8 = "JAM POT"`
(e) `LET L17 = 38`

The Oric also allows you to define integer variables – stores which can only hold whole numbers – which are distinguished by putting a % sign after the variable name, e.g.

```
A%
```

The proper name for an ordinary number variable is a floating-point variable. We shall not deal with integer variables in this course – if you go on to take M034, Structured Programming in BASIC, when you have completed this course, you will come across the best ways to use integer and floating-point variables.

3.2 More about strings

'How long is a piece of string?' is a pertinent question here. In other words, how many characters can be input, stored or output as a single group? In Oric BASIC, a string variable may contain up to 255 characters, provided there is sufficient memory available in which to store the strings. But initially in this course we will assume that a store location for strings will hold up to 40 characters, and that this restriction will apply to inputting and outputting strings. So you must now think of a string memory location, not just as a labelled pigeon-hole, but as a location with 40 sub-divisions, as shown in Figure 2.

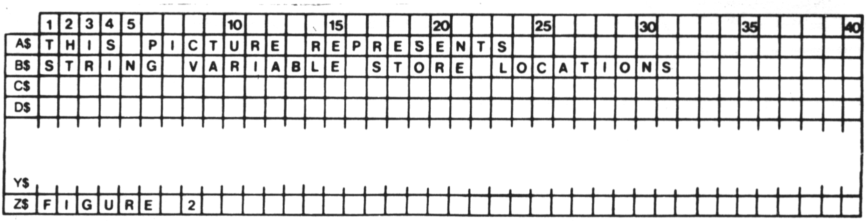


Figure 2 Size of string store locations

We have been discussing strings as if they were new but you have met them before in Unit 1. There we used the PRINT statement to output messages which were enclosed in quotation marks. We implied that the string enclosed in quotation marks was output literally character by character. Then we saw in Unit 2 how the commas between the elements of a PRINT statement caused the strings to be spaced out across the screen or printer.

3.3 PRINT ...;...

From what we have covered so far you will realise that the layout of information on the screen is very important. This is just as true for strings as it is for numbers.

When handling textual information, i.e. strings of characters in the form of words or codes, we want the strings to be printed as in a sentence and not spaced out across the screen in print zones. The PRINT ...;... statement achieves this effect for us. PRINT H$; T$ will take the characters in store location T$ and print them on the left-hand side of the screen followed immediately by the characters from location H$.

In the next few pages we are going to simulate a data recording service of the not too distant future and use it to demonstrate the input and output of strings. Let's start by writing a program which simulates a telephone answering service.

```
 10 REM ** TELE ANSWER **
 20 CLS
 30 PRINT "Hello"
 40 PRINT "Please state your phone number"
 50 INPUT T$
 60 PRINT ←——————————This prints a blank line
 70 PRINT "Hello",T$
 80 PRINT
 90 PRINT "Hello";T$
100 PRINT
110 PRINT "Hello ";T$
120 PRINT
130 PRINT "Hello";" ";T$
140 END
```

Program 1 Printing strings

In common with some forms of BASIC, Oric BASIC allows you to type the ? character instead of the word PRINT, to save typing time and effort. When you subsequently relist the program the word PRINT will replace the ? in each appropriate case.

To help you analyse this program we have put below a 'trace' at the side of a typical run. The trace indicates which line in the program generates which line in the output.

```
RUN                                                 Trace
Hello                                              ...30
Please state your phone number                     ...40
? 58354                                            ...50
Hello      58354                                   ...70
Hello58354                                         ...90
Hello 58354                                        ...110
Hello 58354                                        ...130

Ready
```

Comments

`Trace 50` `INPUT T$` generated ?. Our response was `58354` which the computer treats as a string and not as a number. (If we had written `INPUT T`, then `58354` would be treated as a number.)

`Trace 70` The `PRINT ...,...` on line 70 prints the 5 of the telephone number further across the output line, whereas

`Trace 90` The `PRINT ...;...` of line 90 prints the 5 immediately adjacent to the o of `Hello`.

Neither way is satisfactory, but:

`Trace 110`
`Trace 130` Lines 110 and 130 show alternative ways of introducing the required spaces, either by printing `HELLO`▽ (as in 110) or by inserting the string ▽ in its own right into the output statement (as in 130). (▽ means a space.)

[K] Program 1.

SAQ 3

Study this program and work out what the print output will be. Write down the output in the grid below.

```
  5 REM ** PRINT LAYOUT EXAMPLES **
 10 CLS
 20 PRINT "Print Layout"
 30 LET B$="BASIC"
 40 LET C$="Course"
 50 PRINT
 60 PRINT B$,C$
 70 PRINT
 80 PRINT B$;C$
 90 PRINT
100 PRINT B$;" ";C$
110 END
```

Program 2

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1

SAQ 4

Write a program which would print out the following:

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
B	A	S	I	C				B	A	S	I	C								
B	A	S	I	C	B	A	S	I	C	B	A	S	I	C	B	A	S	I	C	
B	A	S	I	C		B	A	S	I	C		B	A	S	I	C				

3.4 INPUT "...";...

We have used the `PRINT "..."` as a prompt for an input statement. In BASIC we can combine these two into one statement. Thus in the above program, we could replace:

```
40 PRINT "Please state your phone number"
50 INPUT T$
```

with

```
40 INPUT "Please state your phone number  ";T$
```

The INPUT statement will always generate a ? however, so in the next program we form the prompt into a direct question.

The next program demonstrates various `PRINT` effects. Since we need to use "`Hello`" several times we first store it in location H$ at the start of the program.

No question mark here

```
 10 REM ** TELE ANSWER **
 15 CLS
 20 LET H$="Hello"
 30 PRINT H$
 40 INPUT "What is your phone no. ";T$
 50 PRINT
 60 PRINT "Your phone number is ";T$
 70 PRINT
 80 PRINT H$;T$
 90 PRINT H$;T$
100 PRINT
110 PRINT H$;" ";T$
120 END
```

Program 3 The computer asks the questions

Computer response to line 40

```
RUN
Hello
What is your phone no.? 58354
Your phone number is 58354

Hello         58354
Hello58354
Hello 58354

Ready
```

[K] Program 3.

SAQ 5

What would appear on the screen when the following program was run assuming your name is John Smith and your age 45?

```
 5 REM ** SAQ 5 **
10 LET T$ = "Thank you"
15 CLS
20 INPUT "What is your name ";N$
30 PRINT
40 PRINT
50 PRINT "Your name is ";N$
60 PRINT
70 PRINT "What is your age";A$
80 PRINT
```

```
 90 PRINT "You are ";A$;" years old"
100 PRINT
110 PRINT
120 PRINT T$;"   ";N$;"    ";A$
130 END
```

Program 4

3.5 Numbers and strings in PRINT statements

We could have entered the telephone number of the previous program into a numeric store location. We would of course soon run into problems if the number were too long, or contained spaces (e.g. 01 693 4539). Let's compare how BASIC would output this data from numeric and string store locations.

In this program note how we use the string of characters in S$ to print a scale across the output page.

```
 10 REM ** TELE ANSWER **
 15 CLS
 20 H$ = "Hello"
 30 S$ = "1234567890123456789012345"
 40 PRINT H$
 50 INPUT  "What is your phone no. ";T$
 60 PRINT
 70 PRINT
 80 PRINT "Your phone number is ";T$
 90 INPUT "Please type it in again";T
100 PRINT
110 PRINT T
120 PRINT
130 PRINT S$
140 PRINT H$,T$
150 PRINT H$,T
160 PRINT S$
170 PRINT H$;T$
180 PRINT H$;T
190 PRINT S
200 PRINT H$;" ";T$
210 PRINT H$;" ";T
220 END
```

Program 5 Printing strings and numbers

```
RUN                                                       Trace
Hello
What is your phone no.? 58354
Your phone number is 58354
Please type it in again?  58354

58354

1234567890123456789012345 130   s$ numbers each print position
Hello          58354                across the page.
Hello          58354
1234567890123456789012345 160
Hello 58354
Hello 58354
1234567890123456789012345 190
Hello 58354
Hello 58354
Ready
```

```
RUN
Hello
What is your phone no.?-9637
Your phone number is -9637

Please type it in again? -9637

-9637

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 130
H e l l o           - 9 6 3 7
H e l l o           - 9 6 3 7
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 160
H e l l o - 9 6 3 7
H e l l o - 9 6 3 7
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 190
H e l l o  -9 6 3 7
H e l l o  -9 6 3 7

Ready
```

note the effect when T$ and T both hold −9637

Ⓚ Program 5.

SAQ 6
Write a program to input your name as a string and your age as a number and to output the message, '`My name is .... and I am ... years old`', filling in the blanks with the relevant information using normal spacing.

Data recording service
The following is a further example of how print layout is achieved in BASIC. We can imagine that in the not too far distant future our TV set, telephone and computer will be linked together as an *intelligent* terminal. On seeing an attractive advertisement we may *dial* a number and the following dialogue might ensue.

```
                                                     Trace
Hello ...                                              25
This is a data recording service                       30

Please enter details requested                         50

Your name? C.A. Smith                                  70
Your telephone number?  23685                          80
Number or name of house?  77                           90
Road?  Chalmers Road                                  100
Town or city?  Worthing                               110
Your Postal Code?  BR7 9QY                            120

Thank you for your enquiry

Your personal details have been
recorded as:
Name C.A. Smith Phone 23685
Address 77 Chalmers Road
        Worthing BR7 9QY

Details of our services and
products will be sent to you.
Your personal details will
remain confidential.

Ready
```

(Of course, '`will be sent to you`' will eventually be '`will now be output to your terminal`' as electronic mail replaces paper letters, and then the only

details needed to be input would be a subscriber code.) The simulated dialogue was achieved by the following program:

```
 10 REM ** THIS IS A DATA RECORDING SERVICE **
 15 REM ** DATA INPUT **
 20 CLS
 25 PRINT "Hello"
 30 PRINT "This is a data recording service"
 40 PRINT
 50 PRINT "Please enter details requested"
 60 PRINT
 70 INPUT "Your name ";N$
 80 INPUT "Your telephone number ",T$
 90 INPUT "Number or name of house";H$
100 INPUT "Road";R$
110 INPUT "Town or city";C$
120 INPUT "Your Postal Code";P$
130 PRINT
140 PRINT
150 PRINT
160 REM ** END OF DATA INPUT **
170 REM ** ENQUIRY OUTPUT **
180 PRINT "Thank you for your enquiry"
190 PRINT
200 PRINT "Your personal details have been"
205 PRINT "recorded as:"
210 PRINT "Name ";N$;" ";
220 PRINT "Phone ";T$
230 PRINT "Address ";H$;" ";R$
240 PRINT "        ";C$;" ";P$
250 PRINT
260 PRINT
270 PRINT "Details of our services and products"
280 PRINT "will be sent to you."
290 PRINT "Your personal details will remain"
300 PRINT "confidential."
310 REM ******************************
320 END
```

Program 6 Data recording service

[K] Program 6.

3.6 Standard letters

A data recording service such as we have just looked at may be in the future, but standard personalised letters are with us now. Such a letter would be composed on a word-processor, but if your micro does not have word processing facilities available, you could achieve modest results using BASIC. Your own choice of letter will be left to you in Exercise 2.

Example 1
A bank recruiting office receives many enquiries about employment. Its policy is to interview suitable applicants initially at its local branch. A stereotyped letter is sent from the recruiting office to each applicant containing individual details of the proposed interview. Devise a BASIC program to write such a letter.

Solution
The following program would do this job.

For an explanation of the meaning of the INPUT strings `A$` to `G$` refer to the REM statements in the program.

```
 10 REM ** LETTER WRITER **
 12 CLS
 15 REM ** APPLICANT'S NAME **
 20 INPUT A$
 25 REM ** APPLICATION LETTER DATE **
 30 INPUT B$
 35 REM ** INTERVIEWER'S NAME **
 40 INPUT C$
 45 REM ** TIME OF INTERVIEW **
 50 INPUT D$
 55 REM ** DATE OF INTERVIEW **
 60 INPUT E$
 65 REM ** INTERVIEW LOCATION **
 70 INPUT F$
 75 REM ** NAME OF EMPLOYEE REPLYING **
 80 INPUT G$
 90 REM ** END OF INPUT **
100 REM ** OUTPUT **
110 PRINT
120 PRINT
130 PRINT
140 PRINT "Dear ";A$;","
150 PRINT
160 PRINT "Thank you for your letter of the"
170 PRINT B$;".";"  We invite you to"
180 PRINT "attend for a job interview with"
190 PRINT C$;" at ";D$;" on the"
200 PRINT E$;" at our office, the"
210 PRINT F$;" Branch."
220 PRINT
230 PRINT "Yours sincerely,"
240 PRINT
250 PRINT
260 PRINT
270 PRINT G$
280 END
```

Program 7 Bank interview letter

This would result in the following run:

```
?Miss Jones
?13th October
?Mr. Fellows
?10.00 am
?20th October
?High St. Sidcup
?C.A. Sidwell

Ready

Dear Miss Jones,

Thank you for your letter of the
13th October.  We invite you to
attend for a job interview with
Mr. Fellows at 10.00 am on the
20th October at our office, the
High St. Sidcup Branch.

Yours sincerely,

C.A. Sidwell
```

The user of the program might well find it difficult to use, since all he gets is a series of prompts ?. He might, therefore, make a skeletal aide-memoire to remind him of the structure of the letter.

```
A$
B$
C$
D$
E$
F$
G$

Dear A$,

Thank you for your letter of the
B$.  We invite you to
attend for a job interview with
C$ at D$ on the
E$ at our office, the
F$ Branch.

Yours sincerely,

G$
```

[K] Program 7.

Exercise 1
An estate agent periodically sends out a letter to check whether clients on his books are still looking for a property, and that his details of their requirements (e.g. type of property, price range, etc.) are correct. Devise a BASIC program to write such a letter.

Exercise 2
We all write letters requesting things, e.g. details of a product, a service, a holiday, a job, etc. Devise a BASIC program to write a letter which will cover as wide a range of applications as possible, leaving you to fill in only the particular details of each enquiry.

3.7 Patterns, files, READ with DATA

READ with DATA
So far in this course we have entered data at the keyboard during the execution of a program as a response to an `INPUT` statement. Another way of introducing data into a program is to store it in `DATA` statements within the program itself and then to `READ` the items into the program from the `DATA` statements as required. Usually, data is stored at the end of a program, or program segment.

Every time the computer comes to a `READ` statement it takes the next item of data from the `DATA` queue and places it in the location specified in the `READ` statement. For a `READ` statement to be executed there must be a corresponding item of `DATA` available in the `DATA` queue.

Thus in this program the following happens.

```
  5 REM ** READ **
 10 CLS
 20 READ A$
 30 READ B$
 40 READ C$
100 DATA "Tom","Dick"
110 DATA "Harry"
```

Program 8

The Oric requires "   " around strings in DATA statements if leading spaces are required.

At 20, READ tells the computer to take the first item of DATA and put it in location A$. The first DATA item occurs in line 100 and is Tom, so Tom goes into location A$. At the next READ statement (30), the computer takes the next DATA item which is Dick, and so on. You can check this by adding

```
50 REM ** CHECK BY PRINTING **
60 PRINT A$
70 PRINT B$
80 PRINT C$
```

into the program.

```
RUN
Tom
Dick
Harry

Ready
```

SAQ 7

What is wrong with this program segment?

```
5 REM ** SAQ 7 **
10 CLS
20 READ A$
30 READ B
40 READ C
50 READ D$
60 DATA Paul,Mary,63
```

Program 9

A more complex example is shown by this program segment:

```
30 READ N$,T$

120 READ C$

150 READ M$

160 GOTO 30

200 DATA Benny,Copper,Draper
```

```
210 DATA Eddie,Gwynne


220 DATA Hettie


230 DATA Morley,Prosser,Smythe,Weeks


240 DATA Wilson,Wright
```

Here is what happens:

First time round loop

N$	reads Benny
T$	reads Copper
C$	reads Draper
M$	reads Eddie

Stores at end of first time round

N$	Benny
T$	Copper
C$	Draper
M$	Eddie

Second time round loop (i.e. next unread item of DATA)

N$	reads Gwynne
T$	reads Hettie
C$	reads Morley
M$	reads Prosser

Stores at end of second time round

N$	Gwynne
T$	Hettie
C$	Morley
M$	Prosser

SAQ 8

What will be the final state of the stores when all the data has been read?

SAQ 9

What would the contents of locations A$, B$ and C$ be after this program segment has read all the DATA items?

```
 5 REM ** SAQ 9 **
10 CLS
15 READ A$
20 REM ** BRANCH BACK **
25 READ B$
30 READ C$
40 GOTO 20
45 REM ***********************
50 DATA Tinker,Tailor,Soldier
60 DATA Sailor,Rich Man
```

Program 10

RESTORE

BASIC also has a further reading instruction, RESTORE. When this is encountered, no matter how far down the DATA list the program has READ, the next READ instruction will go back and read the first item in the DATA list once more, and then continue reading through the list, i.e. the second item, third item, and so on. This can be very useful if you wish to access the same data more than once in the course of a program, but only wish to key in the data once. Care must be taken that the number of READs and the number of DATA items to read tie up – otherwise a message OUT OF DATA, followed by the offending line number, will appear, stopping the program run.

Files and records

Quite often we want to record data which is of one kind, i.e. it makes up a file of information. A telephone directory is a good example of what in data processing is called a file, that is a collection of similar records. Each record has the form

Name	Address	Telephone number

and is said to consist of a number of fields – in this case three: name, address and telephone number. A record, then, is a collection of fields and a file is a collection of records. A telephone directory is arranged in alphabetical order of surnames which gives it a simple structure.

Comparing strings

We may wish to compare strings. Suppose, for example, we have a personal telephone directory in our microcomputer and we wish to find out whether SMITH is in our record. The computer will have to compare the string "SMITH" against all the strings in the name field of our directory. It can do this very easily because each letter is represented inside the computer by a binary code. Thus

A is 100 0001
B is 100 0010

and so on. (See the appendix to this unit for a full list of binary codes.)
So words placed in alphabetical order on paper will be represented in the computer by codes in numerical order.

Thus if

A$ = cat
B$ = dog
C$ = cat
D$ = fish
E$ = cats

A$ = C$

But B$ > A$ (it is further on in the alphabet)
and E$ > A$ (the extra s on cat puts it after cat in alphabetical order.)

We shall now use this facility in our examples.

Example 2
Set up a data file of names and associated telephone numbers. Write a BASIC program to search through the file to find a particular name, and if found, then output the associated telephone number.

Solution
We could attempt a descriptive algorithm as follows:

1. Start.
2. Input query name.
3. Read next record of the data file (i.e. name and number).
4. If the end of the file has been reached (data name = "zzzz") then output message 'not found in file' and go to 7 otherwise carry on to 5.
5. If query name = data name then output name and number and go to 7 otherwise carry on to 6.
6. Return to 3 for next record.
7. Stop.

However, BASIC does not generally allow statements as complicated as 4 and 5, and so we have to split up these statements as shown in the next algorithm:

1. Start.
2. Input query name.
3. Read next record from data file.
4. If the end of file has been reached then go to 7 otherwise carry on to 5.
5. If query name = data name then go to 9 otherwise carry on to 6.
6. Return to 3 for next record.
7. Output message 'not in file'.
8. Stop.
9. Output name and number.
10. Stop.

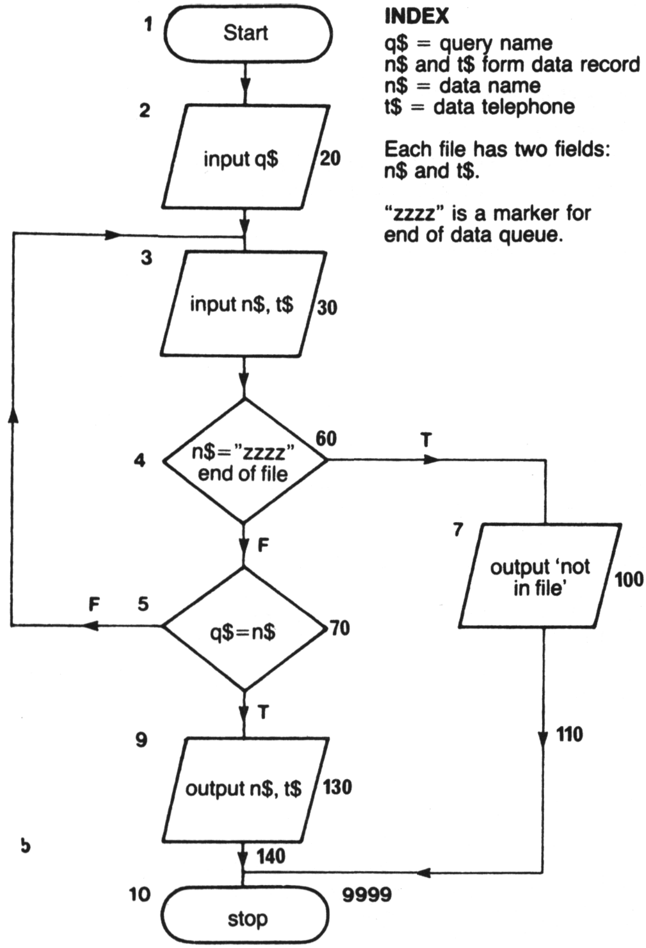


Figure 3 Searching a telephone directory

Now each record contains two fields on this occasion: name and number.

	Field 1	Field 2
	Name N$	Telephone Number T$
e.g.	BENNY	1234

so each DATA item must contain information for each field. Thus the next line will be:

```
READ N$, T$
```

and the DATA lines are of the form:

```
DATA Benny, 1234
```

Notice that the end-of-file number is a DATA item (line 310) so it has to have data to fill T$ as well as N$. Without this an error message would appear on the screen. So we write

```
DATA ZZZZ, end of file
```

and not just

```
DATA ZZZZ
```

```
  10 REM ** TELEPHONE DIRECTORY **
  15 CLS
  20 PRINT "Telephone Directory"
  25 PRINT
  30 INPUT "Surname of person sought ";Q$
  40 REM ** READ NEXT ENTRY **
  50 READ N$,T$
  60 IF N$ = "ZZZZ" THEN GOTO 90
  70 IF Q$=N$ THEN GOTO 120
  80 GOTO 40
  90 REM ** NAME NOT LISTED **
 100 PRINT Q$;" is not in the file"
 110 GOTO 9990
 120 REM ** NAME HAS BEEN FOUND **
 130 PRINT Q$;"'s number is ";T$
 140 GOTO 9990
 190 REM ** DATA LIST **
 200 DATA Benny,1234
 210 DATA Copper,9823
 220 DATA Draper,1850
 230 DATA Eddie,7294
 240 DATA Gwynne,5821
 250 DATA Hettie,4539
 260 DATA Morley,7830
 270 DATA Prosser,1383
 280 DATA Smythe,1147
 290 DATA Weeks,5529
 300 DATA Wilson,9936
 310 DATA ZZZZ,end of file
9990 REM ** ALL FINISHED **
9999 END
```

Program 11 Telephone Directory

Typical runs

```
RUN
Telephone Directory

Surname of person sought? Eddie

Eddie's number is 7294

Ready
```

```
RUN
Telephone Directory

Surname of person sought? Browne

Browne is not in the file

Ready

RUN
Telephone Directory

Surname of person sought? Weeks

Weeks's number is 5529

Ready

RUN
Telephone Directory

Surname of person sought? weeks

weeks is not in the file

Ready

RUN

Telephone Directory

Surname of person sought? Week

Week is not in the file

Ready
```

In the fourth run we entered the name "weeks" whereas the name "Weeks" was actually in the file, and in run five, we entered the name "Week". The computer compares these patterns of characters and finds them unequal, so particular care must be taken when accessing data to ensure that spelling and capital letters used are identical. If we were looking through a telephone directory we might realise that we were really looking for Weeks rather than Week, in the case of run 5. We could, of course, rerun the program with a variety of spellings of a name, if we were in doubt.

[K] Program 11.

SAQ 10
What changes would you have to make to Program 11 to input a person's telephone number and output the subscriber's name, or 'is not in list'?

3.8 Sorting

You will have noticed that the telephone directory data in Example 2 was in alphabetical order as you would expect. It would be difficult for the user in normal practice if this were not so. However, our solution to this searching problem did not use this information; we just searched through the data file record by record until we found the name, or reached the end of the file. Our algorithm would have worked equally well had the data not been in alphabetical order. We shall spend some time later in the course sorting and searching data, at which point you will realise the advantages of sorting data into alphabetical order.

Let's make a modest start with this problem.

Example 3
Write a BASIC program to enter two names into the computer and output that name which would come first in alphabetical order.

Solution
Descriptive algorithm

1. Start.
2. Input first name.
3. Input second name.
4. If first name < second name then 7 otherwise carry on to 5
5. Output second name.
6. Go to 8.
7. Output first name.
8. Stop.

An outline flowchart for the solution of this problem is shown in Figure 4.

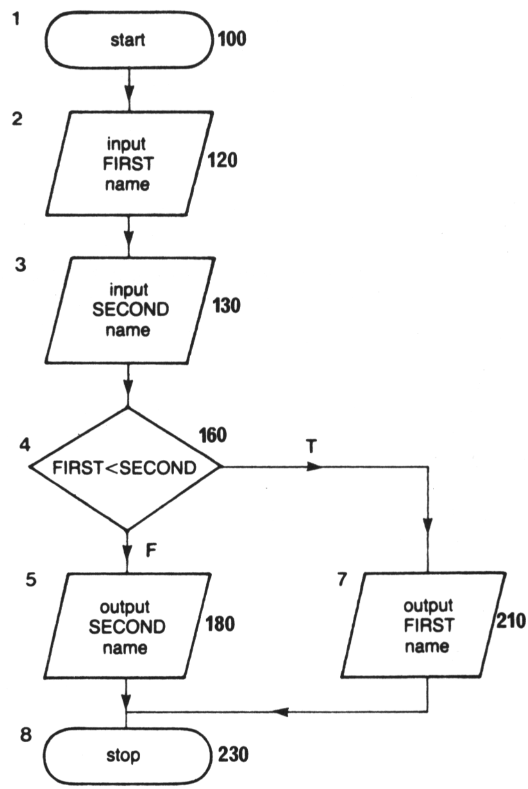


Figure 4 Finding the first of two in alphabetical order

```
110 REM **FIRST IN ALPHA-ORDER**
110 CLS
115 PRINT "Alpha Order Program"
120 INPUT "First Name";A$
130 INPUT "Second Name";B$
140 PRINT
150 REM **TESTING AND OUTPUTS**
160 IF A$ < B$ THEN GOTO 200
170 REM **B$ IS FIRST**
180 PRINT "First in alpha-order is ";B$
190 GOTO 220
200 REM **A$ IS FIRST**
210 PRINT "First in alpha-order is ";A$
240 REM **END NOW**
250 END
```

Program 12

Typical runs

```
RUN
Alpha Order Program
First name? Brown
Second name? Smith

First in alpha-order is Brown

Ready

RUN
Alpha Order Program
First name? Smith
Second name? Brown

First in alpha-order is Brown

Ready
```

[K] Program 12.

The three card trick

Suppose now that we wanted to input three names and output the name that comes first in alphabetical order. A standard solution to this would be to follow the approach in Figure 5.

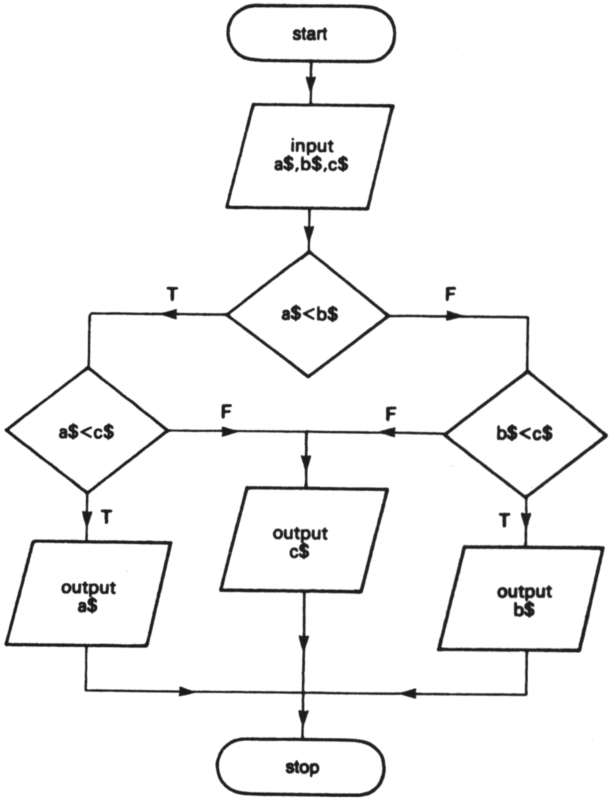


Figure 5 Finding the first of three in alphabetical order

When asked to solve this problem most students present an answer similar to the algorithm in Figure 5. It is a perfectly good solution, but it bodes ill for the future. Future trouble stems from the fact that we have had to utilise three decision and three output functions. This method would overtax our patience and ingenuity if we tried to repeat it for four, five ... let alone ten names. The fundamental problem is allowing each variable to retain its own individual storage location, and how we are best able to label that location.

A simpler method of solving this problem is to input the names one by one and to store lowest-so-far in A$. The program retains only lowest-so-far, destroying all the other discarded data. In the next exercise we suggest you try this approach to solving the problem.

Exercise 3
Write a BASIC program to input three names and output that name which would come first in alphabetical order using the method discussed in the last few lines.

Exercise 4

A data file of European countries and their capital cities is suggested below. Write a BASIC program to use this file as the basis of a quiz with the user, presenting him with the country and asking him to name the capital city. Respond to his input by telling him whether he is correct or not, and in the latter case by giving him the correct answer.

```
210 DATA France,Paris
220 DATA West Germany,Bonn
230 DATA The Netherlands,The Hague
240 DATA Poland,Warsaw
250 DATA Italy,Rome
260 DATA Spain,Madrid
270 DATA Portugal,Lisbon
280 DATA Hungary,Budapest
290 DATA Denmark,Copenhagen
300 DATA Norway,Oslo
310 DATA ZZZZ,end of file
```

Assignment 3

1. Compose a descriptive algorithm and draw a flowchart to accompany the BASIC program in Program 15.

2. Modify your program for Exercise 4 to count the numbers of correct and incorrect responses, and to give a summary of the marks at the end of the quiz.

3. Devise an algorithm and write a BASIC program to do the following task. Store several words as individual letters in DATA statements, e.g. the words 'algorithm' and 'flowchart' could be stored:

```
900 DATA a,l,g,o,r,i,t,h,m
910 DATA f,l,o,w,c,h,a,r,t
```

Count the number of vowels and consonants contained in the words, and output the totals of these counts together with the ratio total number of vowels/total number of consonants.

Objectives of Unit 3

Now that you have completed this unit, check that you are able to use the following in simple programs:

String storage locations ☐
`PRINT ...; ...` ☐
`INPUT "..."; ...` ☐
`READ` ☐
`DATA` (one field only) ☐
`IF A$ = B$ THEN ...` ☐
`DATA` (more than one field) ☐
Simple sorting procedure ☐

Answers to SAQs and Exercises

SAQ 1

Valid string locations: A$, T7$. (It appears that the Oric will also accept 8P$, but we shall not use this format as many BASICs do not accept it).

SAQ 2

(a), (b) and (e) are correct although (e) is not acceptable in minimal BASIC.
(c) is incorrect; M$ is a string location so you need "9583".
(d) is incorrect; K8 is a number store location so the string "JAM POT" cannot be assigned to it.

SAQ 3

```
Print Layout

BASIC      Course

BASICCourse

BASIC Course
```

SAQ 4

```
 5 REM **SAQ 4**
10 CLS
20 LET B$ = "BASIC"
30 PRINT B$,B$
40 PRINT B$;B$;B$;B$
50 PRINT B$;" ";B$;" ";B$
50 END
```

Program 13

```
RUN

BASIC      BASIC
BASICBASICBASICBASIC
BASIC BASIC BASIC

Ready
```

SAQ 5

```
What is your name? JOHN SMITH
Your name is JOHN SMITH
What is your age? 45
You are 45 years old
Thank you   JOHN SMITH   45
```

SAQ 6

```
 5 REM **SAQ 6**
10 CLS
20 INPUT "What is your name";N$
30 INPUT "What is your age";A
40 PRINT
50 PRINT "My name is ";N$;" and I am "
60 PRINT A;" years old."
70 END
```

Program 14

```
RUN
What is your name? Mary
What is your age? 22

My name is Mary and I am
22 years old.

Ready
```

Exercise 1
Exercises 1 and 2 are very similar in nature and an answer for Exercise 1 has not been included.

Exercise 2

```
 10 REM **Enquiry Letter**
 15 CLS
 20 PRINT "Enquiry Letter program"
 30 PRINT
 40 PRINT "Details of addressee"
 50 PRINT
 60 INPUT "Name...";N$
 70 INPUT "Street...";S$
 80 INPUT "Town...";T$
 90 PRINT
100 INPUT "Date for letter";D$
110 PRINT
120 INPUT "Item of interest";I$
130 INPUT "Source of Information";A$
140 INPUT "Date of source";E$
150 WAIT 200
160 CLS
170 REM **Letter production**
180 PRINT N$
190 PRINT S$
200 PRINT T$
210 PRINT
220 PRINT D$
225 PRINT
230 PRINT "Dear Sir,"
240 PRINT
250 PRINT "Will you kindly send me details"
260 PRINT "of ";I$;","
270 PRINT "as itemised in the following:"
280 PRINT A$
290 PRINT "dated ";E$;"."
300 PRINT
310 PRINT
320 PRINT "Yours faithfully,"
330 PRINT
340 PRINT
350 PRINT "O.L. Seymour"
```

Program 15

As previously mentioned, the `CLS` statement clears the screen display, leaving it blank. It does not affect the contents of any variables. It is useful when you have some information to display on the screen, and then wish to display a further screenful of information, without sending the lines of information up the screen.

If we wish to display information for a given amount of time, before we clear the

screen (with CLS) and then display something else, we can use the `WAIT` statement.

```
WAIT N
```

stops the processing and displays the picture on the monitor screen for N periods of 10 milliseconds, where N is a number specified by the programmer. `WAIT (100)` means a one second delay; `WAIT (200)` means a two second delay, and so on.

Typical run

```
RUN
Enquiry Letter Program
Details of addressee
Name...? E.P. Software Ltd.
Street...? Edgware Road
Town...? London

Date for letter? 12th July 1983
Item of interest? business software
Source of information? Modern Computing
Date of source? 10th July


E.P. Software Ltd.
Edgware Road
London

12th July 1983

Dear Sir,

Will you kindly send me details of business
software as itemised in the following:
Modern Computing dated 10th July.

Yours faithfully,

O.L. Seymour
```

SAQ 7

Line 20 will try to read the second `DATA` item which is `Mary`, which is a string, but the location in line 20 is a number location. The computer would stop and indicate a syntax error. To read `"Mary"`, line 20 must be `20 READ B$`.

Line 40 calls for a fourth `DATA` item but there are only three items in line 50.

SAQ 8

`N$` Smythe
`T$` Weeks
`C$` Wilson
`M$` Wright

SAQ 9

A$ Tinker   B$ Sailor   C$ Rich Man

Notice that "Rich Man" is read as one item since there is no comma between the two words. If you try to run this program you will get a message such as 'out of data at line 20'. Why?

SAQ 10

```
  10 REM ** TELEPHONE DIRECTORY **
  15 CLS
  20 PRINT "Telephone Directory"
  25 PRINT
  30 INPUT "Number of person sought ";A$
  40 REM ** READ NEXT ENTRY **
  50 READ N$,T$
  60 IF N$ = "ZZZZ" THEN 90
  70 IF A$ = T$ THEN 120
  80 GOTO 40
  90 REM ** NUMBER NOT LISTED **
 100 PRINT A$;" is not in the file"
 110 GO TO 9990
 120 REM ** NUMBER HAS BEEN FOUND **
 130 PRINT A$;" is the number of ";N$
 140 GO TO 9990
 190 REM ** DATA LIST **
 200 DATA Benny,1234
 210 DATA Copper,9823
 220 DATA Draper,1850
 230 DATA Eddie,7294
 240 DATA Gwynne,5821
 250 DATA Hettie,4539
 260 DATA Morley,7830
 270 DATA Prosser,1383
 280 DATA Smythe,1147
 290 DATA Weeks,5529
 300 DATA Wilson,9936
 310 DATA ZZZZ,end of file
9990 REM ** ALL FINISHED **
9999 END
```

Program 16

Compare this with Program 11.

Exercise 3

A simple method of solving this problem is shown in Program 15. The names are input one by one, and the 'lowest so far' always stored in A$. The program, however, only retains this one item of information; all other data is lost.

```
 10 REM ** FIRST IN ALPHA-ORDER **
 10 CLS
 15 PRINT "Alpha-order Program"
 20 INPUT "First Name ";A$
 25 REM ** NEXT INPUT **
 30 INPUT "Next Name ";B$
 40 IF B$ = "ZZZZ" THEN 100
 50 IF A$ < B$ THEN 70
 60 A$ = B$
 70 REM ** OUTPUT **
 80 PRINT "First so far is  ";A$
 90 GOTO 25
100 REM ** FINAL ANSWER **
110 PRINT
120 PRINT A$;" was overall first."
130 END
```

Program 17

```
RUN
Alpha-order Program
First Name? Tom
First Name? Sid
First so far is Sid
First Name? Joe
First so far is Joe
First Name? Fred
First so far is Fred
Next Name? Bill
First so far is Bill
Next Name? Ron
First so far is Bill
Next Name? Alan
First so far is Alan
Next Name? ZZZZ

Alan was overall first.
```

Exercise 4

```
 10 REM ** EUROPEAN CAPITALS QUIZ **
 15 CLS
 20 PRINT "'Name the Capital' Quiz"
 30 PRINT
 40 REM ** READ NEXT LIST ENTRY **
 50 READ C$,T$
 60 IF C$ = "ZZZZ" THEN 170
 70 PRINT "What is the capital of "
 80 PRINT C$
 90 INPUT "Your answer: ";A$
 95 PRINT A$
100 IF A$ = T$ THEN 140
110 PRINT "No! Sorry, the capital of"
120 PRINT C$;" is ";T$
130 GOTO 30
140 REM ** RIGHT ANSWER **
150 PRINT "Yes! That's right!"
160 GOTO 30
170 REM ** QUIZ FINISHED **
180 PRINT
190 PRINT "That's the end of the quiz."
200 REM ** DATA LIST **
210 DATA France, Paris
220 DATA West Germany, Bonn
230 DATA The Netherlands, The Hague
240 DATA Poland, Warsaw
250 DATA Italy, Rome
260 DATA Spain, Madrid
270 DATA Portugal, Lisbon
280 DATA Hungary, Budapest
290 DATA Denmark, Copenhagen
300 DATA Norway, Oslo
310 DATA ZZZZ, end of file
320 END
```

Program 18

```
RUN
"Name the Capital" Quiz

What is the capital of
France
Your answer? Paris
```

```
Paris
Yes! That's right!

What is the capital of
West Germany
Your answer? Berlin
Berlin
No! Sorry, the capital of
West Germany is Bonn

What is the capital of
The Netherlands
Your answer? Hague
Hague
No! Sorry, the capital of
The Netherlands is The Hague

What is the capital of
Poland
Your answer? Warsaw
Warsaw
Yes! That's right!

What is the capital of
Italy
Your answer? Rome
Rome
Yes! That's right!

What is the capital of
Spain
Your answer? Madrid
Madrid
Yes! That's right!

What is the capital of
Portugal
Your answer? Lisbon
Lisbon
Yes! That's right!

What is the capital of
Hungary
Your answer? Prague
Prague
No! Sorry, the capital of
Hungary is Budapest

What is the capital of
Denmark
Your answer? Copanheen
Copanheen
No! Sorry, the capital of
Denmark is Copenhagen

What is the capital of
Norway
Your answer? Oslo
Oslo
Yes! That's right!

That's the end of the quiz.

Ready
```

Note: There is no need to use the `CLEAR` statement to make space for the `DATA` from a `READ` statement in Oric BASIC, as the `RUN` command clears space automatically for you. If you wish to clear the screen without clearing out all data, use the `CLS` (clear the screen) function.

Appendix

This program prints out all the Oric characters, with their code numbers, based on ASCII (the American Standard Code for Information Interchange), and their binary equivalents.

Line 20 is the start of the FOR ... NEXT ... loop (see Unit 4) which controls which characters are output. There are no characters before 32 (space), which is dealt with separately, so the loop begins at 33 (exclamation mark). It is possible to run straight through all the values, up to 255, but be warned that control codes are held between 126 (swung dash) and 161 (first graphic symbol), so the screen or printer display will go haywire whilst those values are output! The graphic symbols start at 161 and end at 224.

```
 10 REM ** ORIC CHARACTERS (ASCII) **
 11 DIM C$(9)
 12 CLS
 14 PRINT "Code     Char. Binary"
 15 PRINT "32";"     ";"space";"00100000"
 20 FOR A = 33 TO 255
 30 GOSUB 100
 40 PRINT A;
 42 IF A > 99 THEN 46
 44 PRINT " ";
 46 PRINT "    ";CHR$(A);"
 50 FOR J = 2 TO 9
 55 PRINT C$(J);
 60 NEXT J
 62 PRINT CHR$(13)
 65 NEXT A
 70 END
100 REM ** BINARY CONVERSION **
110 B = A
120 FOR I = 1 TO 9
130 C$(I) = "0"
140 NEXT I
150 I = 9
160 IF B / 2 <> (B / 2) THEN C$(I) = "I"
170 B = INT (B / 2)
180 I = I - 1
190 IF I = 1 THEN 210
200 GOTO 160
210 RETURN
```

Program 19

It is necessary to save some space in the computer's memory for storing the ASCII codes as their binary values. We have done this using the DIM (dimension) statement in line 11.

Code	Character	Binary
32	space	00100000
33	!	00100001
34	"	00100010
35	#	00100011
36	$	00100100
37	%	00100101
38	&	00100110
39	'	00100111
40	(	00101000
41	)	00101001
42	*	00101010
43	+	00101011
44	,	00101100
45	-	00101101
46	.	00101110
47	/	00101111
48	0	00110000
49	1	00110001
50	2	00110010
51	3	00110011
52	4	00110100
53	5	00110101
54	6	00110110
55	7	00110111
56	8	00111000
57	9	00111001
58	:	00111010
59	;	00111011
60	<	00111100
61	=	00111101
62	>	00111110
63	?	00111111
64	@	01000000
65	A	01000001
66	B	01000010
67	C	01000011
68	D	01000100
69	E	01000101
70	F	01000110
71	G	01000111
72	H	01001000
73	I	01001001
74	J	01001010
75	K	01001011
76	L	01001100
77	M	01001101
78	N	01001110
79	O	01001111
80	P	01010000
81	Q	01010001
82	R	01010010
83	S	01010011
84	T	01010100
85	U	01010101
86	V	01010110
87	W	01010111
88	X	01011000
89	Y	01011001
90	Z	01011010
91	[	01011011
92	\	01011100
93	]	01011101
94	↑	01011110

95	£	01011111
96	©	01100000
97	a	01100001
98	b	01100010
99	c	01100011
100	d	01100100
101	e	01100101
102	f	01100110
103	g	01100111
104	h	01101000
105	i	01101001
106	j	01101010
107	k	01101011
108	l	01101100
109	m	01101101
110	n	01101110
111	o	01101111
112	p	01110000
113	q	01110001
114	r	01110010
115	s	01110011
116	t	01110100
117	u	01110101
118	v	01110110
119	w	01110111
120	x	01111000
121	y	01111001
122	z	01111010
123	{	01111011
124	|	01111100
125	}	01111101
126	~	01111110
160		10100000
161		10100001
162		10100010
163		10100011
164		10100100
165		10100101
166		10100110
167		10100111
168		10101000
169		10101001
170		10101010
171		10101011
172		10101100
173		10101101
174		10101110
175		10101111
176		10110000
177		10110001
178		10110010
179		10110011
180		10110100
181		10110101
182		10110110
183		10110111
184		10111000
185		10111001
186		10111010
187		10111011
188		10111100
189		10111101
190		10111110
191		10111111

192	11000000
193	11000001
194	11000010
195	11000011
196	11000100
197	11000101
198	11000110
199	1100111
200	11001000
201	11001001
202	11001010
203	11001011
204	11001100
205	11001101
206	11001110
207	11001111
208	11010000
209	11010001
210	11010010
211	11010011
212	11010100
213	11010101
214	11010110
215	11010111
216	11011000
217	11011001
218	11011010
219	11011011
220	11011100
221	11011101
222	11011110
223	10111111
224	11100000

Comment 1
Eight electronic circuits each in the on (1) or off (0) state may be used to represent the characters shown.

Comment 2
Without knowing anything about binary representation we can still pick out another useful feature of the code. If we read:

A 'one million and one'
B 'one million and ten'
Z 'one million, eleven thousand and ten'

even though this interpretation is incorrect we can still see that the letters are ranked in order A B C ... Y Z. We will find this fact very useful subsequently.

UNIT 4
Lists

4.1 Variables

We have already seen how a memory location may store several different values during the course of a program's execution. Thus the value in a store location may vary during a run, and so we often refer to the location names as variables in a program. Thus the 286 store labels:

```
A, A0, A1, ... A9, B, B0, ... Z8, Z9
```

are called numeric variables, and their 286 counterparts:

```
A$, B$, ... Z$,
```

are called string variables. The store labels are used in expressions in program statements just as mathematicians use variables in equations.

4.2 Lists

Lists and supermarkets seem inextricably linked. We go in with a list of items which we wish to buy, and emerge with the items and a list of prices in the form of a receipt. The list of prices results from the process of transferring the items from the basket to the counter. This is a fairly random process but we could have given the list a more meaningful order in a variety of ways. With a lot of effort we could have taken the items out of the basket in order of price, i.e. the cheapest first, the next cheapest next, and so on with the most expensive last, so that the till roll of prices would be in order of cost. Similarly, we could have taken them out of the basket in order of weight, the lightest first through to the heaviest last; and by so doing impose a completely different order on the receipt list. Being able to relate the position in the list in some way to the value of the item is the most useful feature of lists, as we shall see by the end of this unit.

4.3 List variables

Most of the data that we have considered so far can be classified into sets. We have considered sets of test marks, sets of names and associated telephone numbers, sets of countries and their capital cities. Most data can be classified in some way. If we are collecting data for some purpose, this very purpose gives the set of values common characteristics. There are obvious advantages to naming the storage locations for items in a set of data in a way which emphasises that all the items belong to one set. Even better, it would be useful if the storage location names identified the position of an item within the set.

For example, storage, or variable, notation; this emphasises that the values are in some way associated with each other, and allocates a position within the set. This is achieved in the following way.

Consider a set of marks in a teacher's mark book. They form a natural list and could be allocated storage locations in the following way:

Item	**Storage location symbol**
The first member of the M-list is 42	M(1) = 42
The second member of the M-list is 67	M(2) = 67
The third member of the M-list is 90	M(3) = 90
etc. ...	

M(1), M(2), M(3) are like separate memory locations. You can take any of the 286 store locations and put numbers in brackets after them to make list store locations, e.g.

List name	**List store locations in that list**
M(I)	M(1), M(2), M(3) ...
C$(I)	C$(1), C$(2), C$(3) ...

The character in the brackets (here shown as I) is called the index of the list, and may be any positive integer within the memory capacity of your computer.

String lists

As you can see from the table above, lists can be string lists as well as numeric lists. Thus if you name a list M(I), it is clearly a list of numbers, but M$(I) would be a list of strings, e.g. a list of names could be stored:

Index	Item	Variable name
1	Jones	N$(1)
2	Alan	N$(2)
3	Smith	N$(3)
etc.	etc.	etc.

Figure 1 String list names

Lists and arrays

A table of data, like that shown in Figure 1, is often referred to as an array of data. With the data displayed in rows and columns in this way (indexed by item), a table is often referred to as a two-dimensional array. A list (i.e. just one column of data) is similarly called a one-dimensional array. We will see how BASIC provides for two-dimensional arrays in a later unit.

DIM or how long is a list?

As long as you choose! We can choose a list to be of any desired length, provided that we warn the system first. We do this with a `DIM` statement which appears in the program before the array it refers to.

DIM for numerical arrays

If you want to use an array, M say, to store eight numbers then you announce your intention with the statement:

```
DIM M(8)
```

This tells the computer to reserve memory space for the eight variables M(1), M(2), M(3), ..., M(8)

DIM for string arrays

When you want to reserve space for a string array you have to tell the computer two things: (a) the number of strings you wish to store in the array and (b) the maximum length of any one string. You do this by a DIM statement of the form:

```
DIM N$(10, 15)
```

maximum string length (15)

number of strings (10)

This creates ten storage locations N$(1), N$(2), N$(3), ..., N$(10) each capable of holding a string of 15 characters.

Items and index numbers

The following paper and pencil exercises should reinforce your understanding of what is meant by the terms item and index, and their often only fleeting relationship. They also prepare the ground for the interchange-sort procedure which we will consider in detail later in this unit.

Example 1

Transfer the item of lowest value in the following list to position 1, by comparing in turn each of the values in the remainder of the list with the current value at position 1. Interchange the items if the one in the remainder of the list is lower than that at position 1. (It is easier to do than to describe!)

List: 3, 42, −8, 9, −11

Start		Compare & interchange stages			
position or index	item	1st run c	2nd run c&i	3rd run c	4th run c&i
1	3	3	−8	−8	−11
2	42	42	42	42	42
3	−8	−8	3	3	3
4	9	9	9	9	9
5	−11	−11	−11	−11	−8

Figure 2 Sort to place lowest number first

SAQ 1

Carry out the procedure shown in Example 1 for the following list of numbers: 6,8,4,7,3,9,1.

4.4 List input and output

Before we can manipulate the items in a list we have to get the list of items into the computer, and after processing usually get another list out.

Example 2

Write a BASIC program to input three numbers into a list and output the elements of the list in reverse order.

Solution
We will call the list A(I). The required program is then:

```
 10 REM ** EXAMPLE 2 **
 20 REM ** REVERSING A SHORT LIST **
 25 CLS
 30 DIM A(3)
 40 INPUT A(1)
 50 INPUT A(2)
 60 INPUT A(3)
 70 PRINT
 80 PRINT
 90 PRINT A(3),A(2),A(1)
100 END
```

Program 1 Reversing the order of the list

Typical run

```
RUN
? 29
? 32
? -17

-17   32   29

Ready
```

[K] Program 1.

We have done as requested in the question, but have not made a significant advance in programming technique since we could have done the job with the techniques of earlier units simply by calling the variables P, Q and R and outputting them as R, Q and P. To do the job better we need to count the list as it is input so that we can use the counter in reverse order when we output the list. The next example adds this refinement.

Counting a list
Example 3
Write a program to input five numbers into a list, to display the items of the list and its index in the form of a table, and then output the elements of the list in reverse order.

Solution
As listed in the question, there are three main parts to the solution and we can display these in flowchart form as in Figure 3.

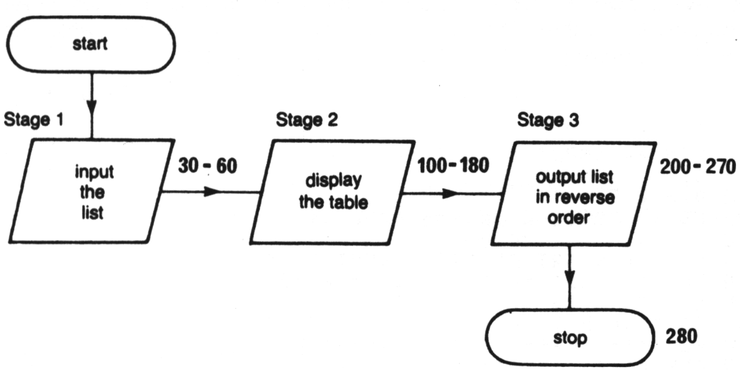


Figure 3 Stages in solving Example 3

Stage 1 We use the variable C to count the elements of the list on input, and to act as an index for the L-list.

```
10 REM ** INPUT A LIST **
15 CLS
20 DIM L(5)
30 C = 1
35 REM ** START OF LIST INPUT LOOP **
40 INPUT "What is the next number ";L(C)
50 C = C + 1
60 IF C <= 5 THEN 35
```

Program 2 Counting a list on entry

Stage 2 The table will have the form you met in the answer to SAQ 1. A simple `PRINT ...,...` will suffice to display the table.

```
100 REM ** DISPLAY THE TABLE **
110 PRINT
120 PRINT
130 PRINT "Index",,"Item"
140 C = 1
150 REM ** START OF TABLE PRINT LOOP ** <---prints the table
160 PRINT C,,,L(C)
170 LET C = C + 1
180 IF C <= 5 THEN 150
```

Program 3 Printing the list in input order

Stage 3 Now we make C count from 5 down to 1 in order to print the list in reverse order.

```
200 REM ** OUTPUT LIST IN REVERSE **
210 PRINT
220 PRINT
230 LET C = 5
240 REM ** REVERSE PRINT LOOP **
250 PRINT L(C)<------------------ prints list in reverse order
260 LET C = C - 1
270 IF C >= 1 THEN 240
```

Program 4 Printing in reverse order

We have shown the three program modules that provide the solution. All we have to do now is put them together as follows. The few changes (which don't affect what the program does) are explained in the comments.

Changes

```
10 REM ** INPUT A LIST **
15 CLS
20 DIM L(5)
25 PRINT "List handling program"
30 C = 1
35 REM ** START OF LIST INPUT LOOP **
40 INPUT "What is the next number ";L(C)
50 LET C = C + 1
60 IF C <= 5 THEN 35
70 REM ************************ ]  REM statements to help
80 REM ************************ ]--reader see the divisions in
                                     the program
```

```
100 REM ** DISPLAY THE TABLE **
110 PRINT
120 PRINT
130 PRINT "Index",,"Item"
140 D = 1
150 REM ** START OF TABLE PRINT **
160 PRINT D,,,L(D)
170 D = D + 1
180 IF D <= 5 THEN 150
190 REM ***********************
195 REM ***********************
200 REM ** OUTPUT LIST IN REVERSE **
210 PRINT
220 PRINT
230 E = 5
240 REM ** REVERSE PRINT LOOP **
250 PRINT L(E)
260 E = E - 1
270 IF E >= 1 THEN 240
280 END
```

We've used D as an index to remind you that its name doesn't matter – only its value

More REMs

And we've used E here

END added

Program 5 The full reverse list program

```
RUN
List handling program
What is the next number? -8
What is the next number? 15
What is the next number? 23
What is the next number? -4
What is the next number? 19

Index          Item
1              -8
2              15
3              23
4              -4
5              19

19
-4
23
15
-8

Ready
```

🄺 Program 5.

To solve the problem in Example 3 we've done a lot of programming but the program only works for a list of five numbers. A small reward for a great effort! But if we change statements 20 to 60 which counted the five items, we can make the program accept any number of items so long as we know the number before inputting.

But we need to make `DIM L` depend on the number of items we are inputting. Because the counter C will go up to N + 1 (watch line 60), `DIM L` needs to be N + 1. We also need to change `'5'` to `'N'` in lines 180 and 230.

```
10 REM ** INPUT A LIST OF N ITEMS **
20 CLS
25 PRINT "List handling program"
26 INPUT "How many numbers in the list? ";N
28 DIM L(N + 1)
30 C = 1
```

Any number of items can now be entered into the list L(C)

```
 35 REM ** START OF LIST INPUT LOOP **
 40 INPUT "What is the next number ";L(C)
 50 C = C + 1
 60 IF C <= N THEN 35
 70 REM ***********************
 80 REM ***********************
100 REM ** DISPLAY THE TABLES **
110 PRINT
120 PRINT
130 PRINT "Index",,"Item"
140 D = 1
150 REM ** START OF TABLE PRINT **
160 PRINT D,,,L(D)
170 D = D + 1
180 IF D <= N THEN 150
190 REM ***********************
195 REM ***********************
200 REM ** OUTPUT LIST IN REVERSE **
210 PRINT
220 PRINT
230 E = N
240 REM ** REVERSE PRINT LOOP **
250 PRINT L(E)
260 E = E - 1
270 IF E >= 1 THEN 240
280 END
```

Program 6

(This program is only suitable for use with short lists. Further work on the display is needed for longer lists.)

To be able to input any number of items of data into individual store locations with just half a dozen statements represents a significant improvement in programming technique. But, of course, we are never satisfied! Why should we bother to count the items, especially if the list is long, when we can get the computer to do it for us? The next exercise asks you to do this.

Exercise 1
Write a BASIC program to (a) input a list of numbers of unknown length; terminate the list with the dummy '−9999'. Call this list P(C) and assume that the list will be 30 or fewer items. So `DIM P(31)` will declare the list. Check your answer. (b) Now modify your program to output those items whose index is odd.

4.5 The FOR ... NEXT ... loop

As we have said (repeatedly!) a computer is good at lots of repetitive operations. In order to control these operations we usually have to count them. Since we introduced the idea of counting in Unit 2, we have used the following sequence of statements several times.

```
LET C = 1
activity
LET C = C + 1
IF C <= N THEN return to activity
```

Program 7

These repetitive operations are so important in programming that special provision is made for them. In BASIC this is done by means of the `FOR ... NEXT ...` facility.

The sequence in Program 7 has three elements:

`LET C = 1`	which starts the count
`LET C = C + 1`	which defines the incremental step (1 in this case; 2 in line 270 of the answer to Exercise 1)
`C <= N`	which stops the counting process

The same features occur in the `FOR ... NEXT ...` loop where the above sequence becomes:

```
        start       finish          increment

   FOR C = 1 TO N STEP (1)
  activity
   NEXT C
```

Program 8

Notice that `NEXT C` returns control to the line `FOR C = 1 TO N STEP (1)` without you having to put the line number of `FOR C ...` in the `NEXT C` statement.

Arrays and `FOR ... NEXT` loops were made for each other. Together they form possibly the most potent facility of the BASIC language. Let's look in detail at how these loops work, and then repeat some of our earlier routines with lists using this new facility.

Examples of FOR ... NEXT loops in action

(a)
```
10 FOR I = 4 TO 10 STEP 2
20 PRINT I
30 NEXT I
40 END

RUN

4
6
8
10

Ready
```

(b)
```
10 FOR K = 11 TO 4 STEP -2
20 PRINT K
30 NEXT K
40 END


RUN
11
9
7
5

Ready
```

(c)
```
10 FOR J = -3 TO 10 STEP 3
20 PRINT J
30 NEXT J
40 END

RUN

-3
0
3
6
9
```

(d)
```
10 FOR L=4 TO -5 STEP -2
20 PRINT L
30 NEXT L
40 END


RUN
4
2
0
-2
-4

Ready
```

Programs 9-12

SAQ 2
Write out the lists of numbers printed out by the following loops.

(a)
```
10 FOR E = 1 TO 9 STEP 2
20 PRINT E
30 NEXT E
40 END
```

(b)
```
10 FOR F = -30 TO -18 STEP 3
20 PRINT F
30 NEXT F
40 END
```

(c)
```
10 FOR G=8 TO -4 STEP -5
20 PRINT G
30 NEXT G
40 END
```

(d)
```
10 FOR H=-2 TO -11 STEP -4
20 PRINT H
30 NEXT H
40 END
```

Programs 13-16

FOR ... NEXT ... with STEP (1)
The above examples and SAQs had steps of 2, 3, −2, −4, and −5. Quite often, however, we simply want to use a step of 1. When that is the case, you can omit STEP (1) from the statement. Thus

```
FOR C = 1 TO N
activity
NEXT C
```

Program 17

is taken by the computer to mean a step of 1.

Input/output routines with FOR ... NEXT
Routines are made easier by the `FOR ... NEXT` facility. For example, we can rewrite Program 5 using these loops. Notice that at lines 40 and 160 we require a step of 1 so STEP (1) has been omitted.

Comparison with Program 5

```
 10 REM ** INPUT A LIST OF 5 NAMES **
 15 CLS
 20 DIM L$(5)
 25 PRINT "List handling program"
 30 REM ** START OF LIST INPUT LOOP **
 40 FOR C = 1 TO 5
 50 INPUT "What is the next name";L$(C)   Replaces lines 40 to 60
 60 NEXT C
 70 REM ***********************
 80 REM ***********************
100 REM ** DISPLAY THE TABLE **
110 PRINT
120 PRINT
130 PRINT "Index",,"Item"
140 PRINT
150 REM ** START OF TABLE PRINT **
160 FOR D = 1 TO 5
170 PRINT D,L$(D)                         Replaces lines 160 to 180
180 NEXT D
190 REM ***********************
195 REM ***********************
200 REM ** OUTPUT LIST IN REVERSE **
210 PRINT
220 PRINT
```

```
230 REM ** REVERSE PRINT LOOP **
240 FOR E = 5 TO 1 STEP -1
250 PRINT E,,,L$(E)
260 NEXT E
270 END
```

Replaces lines 250 to 270

Program 18 Using `FOR ... NEXT ...` to reverse a list

```
RUN

List handling program
WHAT IS THE NEXT NAME? Dickens
WHAT IS THE NEXT NAME? Hardy
WHAT IS THE NEXT NAME? Snow
WHAT IS THE NEXT NAME? Austen
WHAT IS THE NEXT NAME? Orwell

Index          Item

1              Dickens
2              Hardy
3              Snow
4              Austen
5              Orwell

5              Orwell
4              Austen
3              Snow
2              Hardy
1              Dickens

Ready
```

[K] Program 18.

General FOR ... NEXT loop

The `FOR ... NEXT` loop can be expressed in general terms as

```
FOR I = S TO F STEP (J)
```

activity

```
NEXT I
```

providing that S, F and J are given 'reasonable' values before the loop is executed. Unreasonable values would be something like

```
S = 2, F = 10, and STEP(-3)
```

since you can't get from 2 to 10 in steps of -3 in the normal course of events.

Exercise 2

In Exercise 2 of Unit 2 you wrote a program (Program 19) to calculate the yield on an investment for a period of years to be specified. Rewrite lines 100-170 of the program using a `FOR ... NEXT` loop.

Exercise 3

If you are mathematically inquisitive you might like to try the following which demonstrates the potential of the `FOR ... NEXT` loop. Write a program to tabulate the squares and cubes of the odd integers from 1 to 21 inclusive.

Output display
We always aim at a clear presentation of output data on the screen or printer. The `FOR ... NEXT` facility is used widely in presenting output routines.

Use to skip lines. As we saw in the layout of letters in Unit 3 it is useful to 'print' blank lines. The following routine does this for you.

```
10 REM ** FOR...NEXT **
20 REM ** TO SKIP LINES IN A PRINT ROUTINE **
30 CLS
40 PRINT "Hello"
50 REM ** START OF LOOP **
60 FOR H = 1 TO 10
70 PRINT
80 NEXT H
90 PRINT "Hello from 11 lines below"
100 END
```

Instruction to print 10 blank lines

Program 19

```
RUN
Hello










Hello from 11 lines below
```

[K] Program 19.

Drawing a line. We may wish to print lines across the screen or page, e.g. to separate blocks of data or just to underline. The following routine does this.

```
10 REM ** FOR...NEXT **
20 REM ** TO DRAW LINES **
30 CLS
40 REM ** START OF LOOP **
50 FOR M = 1 TO 36
60 PRINT "*";
70 NEXT M
80 PRINT
90 END
```

Instruction to print * 36 times

Program 20

```
RUN
************************************
```

SAQ 3
Why does line 60 in Program 20 have ; at the end of the line? What happens if line 60 is

```
60 PRINT "*"
```

[K] Program 20

Loops in flowcharts

Loops are so important that they have a special flowchart symbol of their own:

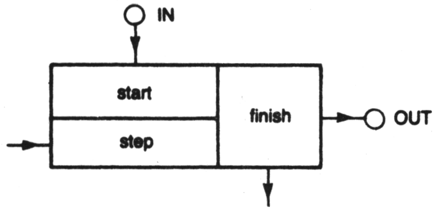


Figure 4a Flowchart symbol for a `FOR ... NEXT ...` **loop**

The floating ends are the connections to the activity:

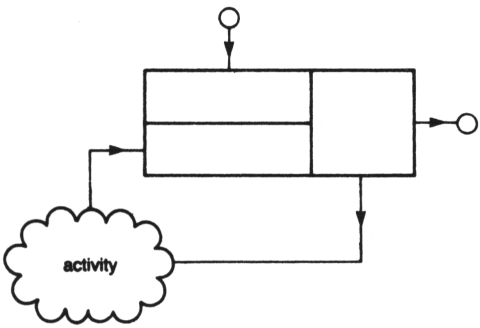


Figure 4b Flowchart symbol's relationship to activity

4.6 Nested loops

Program 20 drew a line of 36 asterisks. We can rewrite the program so that we can specify in line 50 a number of asterisks and so vary the length of the line. So with the program:

```
10 REM **FOR...NEXT**
20 REM **TO DRAW LINES**
30 CLS
35 N=1◄──────────────── Number to go in line 50
40 REM**START OF LOOP**
50 FOR M = 1 TO N
60 PRINT "*";
70 NEXT M
80 PRINT
90 END
```

Program 21

we get

```
RUN
*

Ready
```

To vary the line length we simply key in

```
35 N = required length
```

and key RUN:

```
35 N = 2
RUN
**

Ready

35 N = 3
RUN
***

Ready

35 N = 4
RUN
****

Ready

35 N = 8
RUN
********

Ready

35 N = 16
RUN
****************

Ready

35 N = 32
RUN
********************************
Ready
```

$\boxed{K}$ Program 21.

Nested FOR ... NEXT... loops

You have seen in Program 21 how you can control the effect of the `FOR ... NEXT ...` loop of lines 50 to 70 by changing the value of *N*. We hope by now that the 'obvious' question springs to your mind: 'Why not control the value of *N* by using another `FOR ... NEXT ... loop`?' The following program does just that. The M-loop of lines 50 to 70 is itself controlled by the N-loop of lines 30 to 100. The M-loop is said to be 'nested' within the N-loop.

```
10 REM **NESTED FOR... NEXT**
15 CLS
20 REM **START OF OUTER LOOP (NO. OF ROWS)**
30 FOR N = 1 TO 16
40 REM **START OF INNER LOOP (NO. OF *
       IN EACH ROW)**
50 FOR M = 1 TO N
60 PRINT "*";
70 NEXT M
80 REM **END OF INNER LOOP**
90 PRINT
100 NEXT N
110 REM **END OF OUTER LOOP**
120 END
```

Inner loop; controls number of * in each row

Outer loop; controls the rows

Program 22 Nested loops

```
RUN

*
**
***
****
*****
******
*******
********
*********
**********
***********
************
*************
**************
***************
****************
```

It is important that you understand how this works. For example, when the computer has just finished printing the 9th row of asterisks N will be 9. It leaves the inner loop (line 70) and control goes to the outer loop (line 30). A blank line is printed and N then increases to its next value of 10. Control reverts to the inner loop at line 50. The computer then goes around the inner loop (lines 50 to 70) ten times before exiting again to line 30.

[K] Program 22.

More print patterns from loops

If generating print patterns with loops appeals to you, here is another one, plus two Exercises.

```
 10 REM **NESTED FOR...NEXT loops**
 15 CLS
 20 P = 1
 30 REM ******************************
 40 FOR N = 1 TO 5
 50 P = 2 * P
 60 REM ******************************
 70 FOR M = 1 TO P
 80 PRINT "*";
 90 NEXT M
100 REM ******************************
110 PRINT
120 NEXT N
130 REM ******************************
140 END
```

Multiplies current value of P by 2 for each pass around loop.

Program 23

```
RUN
**                                    P= 2
****                                  P= 4
********                              P= 8
****************                      P=16
********************************      P=32
```

[K] Program 23.

It is easier to see the program's structure at a glance if the text is indented – in this case, the inner loop is moved further to the right than the outer loop. To save memory the Oric removes leading spaces after the first space. To overcome this it is necessary to start the line with a colon (:). We shall not, as a general rule, be indenting programs in the text, but it is as well to know that this facility is available.

```
 10 REM**NESTED FOR... NEXT**
 15 CLS
 20 P = 1
 30 REM ****************************
 40 FOR N = 1 TO 5
 50 : P = 2 * P
 70 :    FOR M = 1 TO P
 80 :       PRINT "*";
 90 :    NEXT M
110 :  PRINT
120 NEXT N
130 REM ****************************
140 END
```

Program 23 (with indented loops)

Exercise 4
Write a program using nested loops to print out the 7, 8 and 9 multiplication tables.

Exercise 5
Write a program using nested loops to print out rectangles of asterisks of dimensions to be chosen by the user.

4.7 Interchanging

We considered the problem of finding the smaller of two numbers in Unit 2, and the smallest of three in Unit 3. In this unit we have done exercises on interchanging items of a list, in preparation for writing an interchange program.

We have been comparing the items of a list with that at position 1, and interchanging if the item in the list is smaller than that at position 1. When you did this in SAQ 1, you did it manually and we have not yet looked at the problems of writing a program to perform the interchange. We can't just say

Copy A into B and then B into A

since the first transaction 'copy A into B' overwrites and destroys what's in B, giving us copies of A in A and in B. Instead we have to put the contents of B away in some safe temporary store before we overwrite B with A. We can show the process diagrammatically:

TEMP ← B B ← A A ← TEMP

Where ← means place the number in the right hand store into the left hand store.

Suppose, for example, you want to sort a list of names N$-list and you want to interchange the first name, N$(1) with some other name N$(K) at position K. This can be done using a temporary store location T$:

T$ ← N$(K)
N$(K) ← N$(1)
N$(1) ← T$

Remember that it is the contents of N$(1) and N$(K) that are being swapped. Suppose N$(1)=FRED and N$(K)=JIM then this is what is happening:

	Store locations		
	N$(1)	N$(K)	T$
start	FRED	JIM	
next stage	FRED	JIM	JIM
next stage	FRED	FRED	JIM
end	JIM	FRED	JIM

(The fact that T$ still has JIM in it doesn't matter: we have achieved the object which is to swap the locations of FRED and JIM.)

Flowchart for name sort

In the number sort (SAQ 1) we wanted to put the lowest number at the top of the list. So we test each name in turn against the one currently at the top of the list and interchange only if the name under test comes before the one at the top of the list. A flowchart for this is:

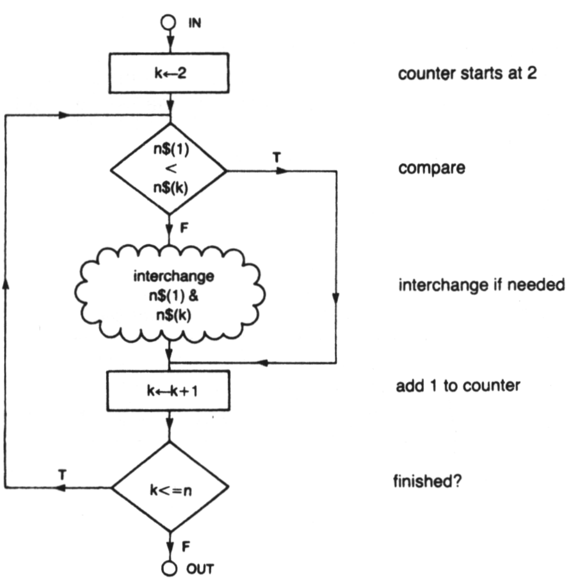


Figure 5a Flowchart for interchange

Or, if we want to use the special flowchart symbol for `FOR ... NEXT ...`, it would look like:

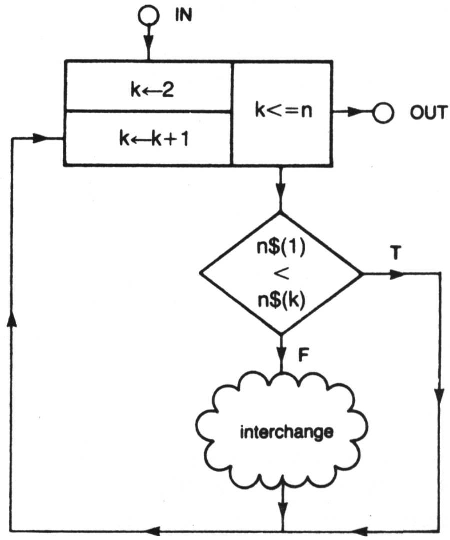


Figure 5b Flowchart for interchange with FOR ... NEXT ... symbol

This new routine can now be used to construct a program.

Example 4

Write a program to enter a list of names of unknown length into an array, print out this list with index in input order. By means of the interchange routine place the name of lowest alphabetic value in position 1 in the list, and output the next list.

Solution

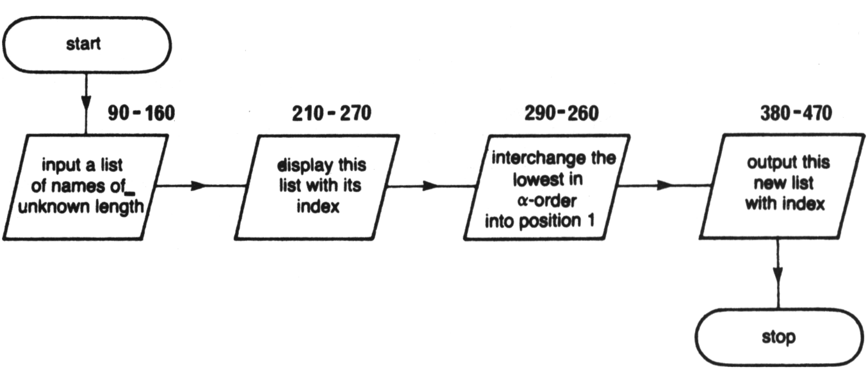


Figure 6 Flowchart for Example 4

Interchange program

```
 10 REM **FIRST IN ALPHA-ORDER**
 15 CLS
 20 PRINT "This program prints out the first"
 24 PRINT "item alphabetically from  a list"
 30 DIM N$(20)
 40 PRINT
 50 PRINT "Enter a list of names one by one"
 60 PRINT "ending the list with zzzz"
 70 PRINT
 80 REM ******************************
 90 REM **INPUTTING LIST**
100 LET I = 1
110 INPUT "Next name";N$(i)
130 IF N$(I)="zzzz" THEN 190                 inputting list
140 I = I + 1
150 REM **GO BACK FOR NEXT NAME**
160 GOTO 110
170 REM ******************************
180 REM **zzzz NOT WANTED IN LIST, SO**      finding number
190 N = I - 1                                of names entered
200 REM ******************************
210 REM **PRINT LIST**
220 PRINT
230 PRINT "Index",,"Item"
240 REM **START OF PRINT LOOP**
250 FOR J = 1 TO N
260 PRINT J,,,N$(j)                          print list
270 NEXT J
280 REM ******************************
290 REM **INTERCHANGE ROUTINE**
300 FOR K = 2 TO N
310 IF N$(1) < N$(K) THEN 350
320 T$ = N$(K)
330 N$(K) = N$(1)                            interchange
340 N$(1) = T$
350 REM **JUMP TO HERE IF ITEMS ALREADY
    IN ORDER**
360 NEXT K
370 REM ******************************
380 REM **OUTPUT INTERCHANGED LIST**
390 PRINT
400 PRINT "List after interchange"
410 PRINT
420 PRINT "Index",,"Item"                    print interchanged order
430 FOR L = 1 TO N
440 PRINT L,,, N$(L)
450 NEXT L
460 REM ******************************
470 END
```

Program 24 Finding the first item in alphabetical order

Interchange program runs

```
RUN

This program prints out the first
item alphabetically from a list


Enter a list of names one by one
ending the list with zzzz
```

```
Next name? Jones
Next name? Price
Next name? Davies
Next name? Evans
Next name? zzzz

Index          Item
1              Jones
2              Price
3              Davies
4              Evans

List after interchange

Index          Item
1              Davies
2              Price
3              Jones
4              Evans


Ready
```

If a print routine is inserted into the interchange procedure as below (lines 342 to 348), then we can look at the effect of each pass round the loop.

```
280 REM ******************************
290 REM **INTERCHANGE ROUTINE**
300 FOR K = 2 TO N
310 IF N$(1) < N$(K) THEN 350
320 T$ = N$(K)
330 N$(K) = N$(1)
340 N$(1) = T$
342 FOR L = 1 TO N  ]
344 PRINT N$(L);" "; ]—— Print routine to observe pass round the loop
346 NEXT L          ]
348 PRINT
350 REM **JUMP TO HERE IF ITEMS ALREADY
    IN ORDER**
360 NEXT K
370 REM ******************************
```

🄺 Program 24 with lines 280 to 370 as above.

Assignment 4

1. It will probably have occurred to you by now that, having placed the item of lowest value into position 1, we could repeat the procedure by placing the item of lowest value in the remainder of the list into position 2, and so on for the rest of the list. The sort of the complete list in this way demands nested `FOR ... NEXT ...` loops.

Modify Program 24 to sort a complete list into alphabetical order.

2. Input a file of names and associated telephone numbers into two lists N$(I) and T$(I) respectively. Use the index I to search through the file to find a particular name, and if found then to output the associated telephone number.

Objectives of Unit 4

Now that you have completed this Unit, check that you are able to write simple programs using:

List store location names
- to input lists □
- to print lists □

Counters to count the number of items in a list □

`FOR ... NEXT ...` loops
- to print a list □
- to input a list □
- to print * layouts □

Nested loops
- to print * layouts □

Interchange routine □

Answers to SAQs and Exercises

SAQ 1

The six stages of the procedure are shown here in the following program run:

```
RUN
Enter a list of names one by one
End the list with zzzz

Next name? 6
Next name? 8
Next name? 4
Next name? 7
Next name? 3
Next name? 9
Next name? 1
Next name? zzzz


Index         Item
1             6
2             8
3             4
4             7
5             3
6             9
7             1

6  8  4  7  3  9  1
4  8  6  7  3  9  1
4  8  6  7  3  9  1
3  8  6  7  4  9  1
3  8  6  7  4  9  1
1  8  6  7  4  9  3

List after interchange
```

Index	Item
1	1
2	8
3	6
4	7
5	4
6	9
7	3

Exercise 1

Notice that we've used lots of `REM` statements to tell you how the program works.

```
 10 REM **INPUT A LIST OF NUMBERS OF UNKNOWN LENGTH**
 15 DIM P(31)
 18 CLS
 20 PRINT "Number list program"
 30 PRINT "Enter the elements of the list"
 40 PRINT "item by item as requested"
 50 PRINT "end list with dummy (-9999)"
 60 PRINT
 70 C = 1
 80 REM **START OF INPUT SEQUENCE**
 90 INPUT "Enter the next number ";P(C)       Input sequence
100 PRINT
110 IF P(C) = -9999 THEN 160
120 C = C + 1
130 GOTO 90
140 REM *****************************
150 REM **'C' COUNTED -9999 AS AN ITEM**
160 REM **INPUT COMPLETED**
170 N = C - 1                                 Taking correct total from counter
180 REM *****************************
190 REM **OUTPUT ITEMS WHOSE INDEX IS ODD**
200 REM **3 = 1 + 2..5 = 3 + 2..**
210 C = 1
220 PRINT
230 PRINT
240 PRINT "Odd index",,"Item"                 Output sequence
250 REM **OUTPUT SEQUENCE**
260 PRINT C,,,P(C)
270 C = C + 2
280 IF C < = n THEN 260
290 REM *****************************
300 END
```

Program 25

```
RUN
Number list program
Enter the elements of the list
item by item as requested
end list with dummy (-9999)

Enter the next number? 42
Enter the next number? -12
Enter the next number? 37
Enter the next number? 92
Enter the next number? 11
Enter the next number? -3
Enter the next number? -9999

Odd index     Item
1             42
3             37
5             11

Ready
```

SAQ 2

(a) 1, 3, 5, 7, 9.
(b) −30, −27, −24, −21, −18.
(c) 8, 3, −2.
(d) −2, −6, −10.

Exercise 2

```
 10 REM **COMPOUND INTEREST**
 15 CLS
 20 PRINT "Bank Account Interest"
 30 PRINT
 40 INPUT "How many years ";N
 50 PRINT
 60 INPUT "What is your deposit (£)";D
 70 PRINT
 80 INPUT "What rate of interest (%)";I
 90 PRINT
100 REM **FOR... NEXT LOOP**
110 FOR C = 1 TO N
120 Y = (I*D)/100
130 PRINT "Year ";C,,"Yield £";Y
140 D = D + Y
150 NEXT C
160 REM ******************************
170 END
```

Program 26

```
RUN
Bank Account Interest

How many years? 5

What is your deposit (£)? 500

What rate of interest (%)? 11.25

Year 1        Yield £56.25
Year 2        Yield £62.578125
Year 3        Yield £69.618641
Year 4        Yield £77.4502075
Year 5        Yield £86.1633559

Ready
```

Exercise 3

```
 10 REM **SQUARES AND CUBES**
 15 CLS
 20 PRINT "Squares and Cubes generation"
 30 PRINT
 40 PRINT "Number----Square----Cube"
 50 REM **START OF LOOP**
 60 FOR I = 1 TO 21 STEP 2
 70 S = I*I
 80 C = I*I*I
 90 PRINT I;
100 PRINT "          ";S;
110 PRINT "          ";C
120 NEXT I
130 END
```

Program 27

```
RUN
Squares and Cubes generation

Number----Square----Cube
1         1           1
3         9           27
5         25           125
7         49           343
9         81           729
11          121           1331
13          169           2197
15          225           3375
17          289           4913
19          361           6859
21          441           9261
```

The display is a bit of a mess, but we will fix that in the next unit with `TAB`.

SAQ 3

`;` suppresses the print return so that the cursor stops after printing `*`. Thus the next `*` will be printed on the same line. Without `;` the asterisks would be printed in one column three print lines deep (which would overflow the screen display).

Exercise 4

```
 10 REM **MULTIPLICATION TABLES**
 15 CLS
 20 PRINT "Multiplication Tables"
 30 REM **'WHICH TABLE' LOOP**
 40 FOR T = 7 TO 9
 50 REM **'GENERATE THE TABLE' LOOP**
 60 FOR K = 1 TO 12
 70 P = K*T
 80 PRINT K;" times ";T;" = ";P
 90 NEXT K
100 REM **END OF INNER LOOP**
110 PRINT
120 NEXT T
130 REM **END OF OUTER LOOP**
140 END
```

Program 28

```
RUN
Multiplication Tables
1 times 7 = 7
2 times 7 = 14
3 times 7 = 21
4 times 7 = 28
5 times 7 = 35
6 times 7 = 42
7 times 7 = 49
8 times 7 = 56
9 times 7 = 63
10 times 7 = 70
11 times 7 = 77
12 times 7 = 84

1 times 8 = 8
2 times 8 = 16
3 times 8 = 24
4 times 8 = 32
5 times 8 = 40
6 times 8 = 48
7 times 8 = 56
8 times 8 = 64
```

```
9 times 8 = 72
10 times 8 = 80
11 times 8 = 88
12 times 8 = 96

1 times 9 = 9
2 times 9 = 18
3 times 9 = 27
4 times 9 = 36
5 times 9 = 45
6 times 9 = 54
7 times 9 = 63
8 times 9 = 72
9 times 9 = 81
10 times 9 = 90
11 times 9 = 99
12 times 9 = 108

Ready
```

We still have a lot to learn about 'tabulation'

Exercise 5

```
 10 REM **RECTANGLE**
 15 CLS
 20 PRINT "Rectangle Program"
 30 INPUT "Length of rectangle";L
 40 INPUT "Width of rectangle";W
 50 REM **OUTER LOOP**
 60 FOR I = 1 TO L
 70 REM **OUTER LOOP**
 80 FOR J = 1 TO W
 90 PRINT "*";
100 NEXT J
110 REM **END OF INNER LOOP**
120 PRINT
130 NEXT I
140 REM **END OF OUTER LOOP**
150 END
```

Program 29

```
RUN
Rectangle Program
Length of rectangle? 7
Width of rectangle? 9
*********
*********
*********
*********
*********
*********
*********

Ready
```

UNIT 5
An end to strings and PRINT

5.1 Introduction

The earlier units were concerned with introducing topics; new ideas came thick and fast. This unit is mainly concerned with strings, but you will meet the `TAB` statement which is an important addition to your printing repertoire. The title of this unit is a slight exaggeration, but by the end of this unit you will have met most of the main string and print functions of the BASIC language.

5.2 Length of a string of characters

We asked the question 'How long is a piece of string?' in Unit 3. At the time it may have seemed a rather facetious question, but the number of characters contained in a particular string storage location is often a vital piece of information. This is especially so if we are trying to use the memory allocation of a particular computer as efficiently as possible.

In BASIC the operation `LEN` (A$) gives the length of A$ as a number of characters. Thus:

```
If A$="Fred"  then LEN (A$) = 4
If A$="I" ,  LEN (A$) = 1
```

SAQ 1
What are the values of the following:

(a) LEN (C$) where C$="Ann" (c) LEN (E$) where E$="72"
(b) LEN (D$) where D$="A" (d) LEN (F$) where F$="CAT 123"

Example 1
Write a BASIC program to input a list of ten words, ending with 'zzzz', and to print out the length of each word.

Solution
We have set up a reading routine to input the words into a string list, W$.

```
 10 REM **LENGTH OF A WORD**
 20 REM ******************************
 30 REM **READ WORDS FROM A DATA LIST
 40 REM ONE BY ONE AND OUTPUT THEIR
 50 REM LENGTHS**
 90 REM ******************************
 95 CLS
100 READ W$
110 IF W$="ZZZZ" THEN 200
120 L=LEN (W$)                          READ, LEN, PRINT cycle
130 PRINT
140 PRINT W$;" has ";L;" letter(s)"
150 GOTO 100
200 END
900 DATA devise,an,algorithm,and
910 DATA write,a,BASIC,program
910 DATA ZZZZ
```

Program 1 Measuring word lengths

```
RUN
devise has 6 letter(s)
an has 2 letter(s)
algorithm has 9 letter(s)
and has 3 letter(s)
write has 5 letter(s)
a has 1 letter(s)
BASIC has 5 letter(s)
program has 7 letter(s)
```

(Result when devise, an, algorithm, and, write, a, BASIC, program, zzzz formed the data.)

[K] Program 1

5.3 Frequency tables

Measuring the frequency with which something occurs is commonly needed in handling numerical information. For example, a knowledge of the frequency with which certain letters occur in normal language usage is an important factor in code-breaking activities. In order to measure frequencies it is useful to be able to use the simple technique used in statisical analysis of tally marks. This first paper and pencil example introduces this.

Tally marks
Example 2
Find the frequency with which each vowel occurs in the following words:

the, horse, stood, still, till, he, had, finished, the, hymn, which, Jude, repeated, under, the, sway, of, a, polytheistic, fancy, that, he, would, never, have, thought, of, humouring, in, broad, daylight

Solution
There are two ways of approaching the problem.

(a) Go through crossing out and counting up all the a's, and then through again counting the number of e's, etc. This would involve five passes through the data for fairly sparse information (i.e. a low hit rate);
(b) Draw up a table as below and take each vowel in sequence:
th~~e~~: put a tally mark in the e row;
h~~o~~rs~~e~~: put a mark in the o row, followed by another in the e row;
st~~oo~~d: put two more marks in the o row.

Vowel	Count	Total count or frequency
a	~~1111~~ 1111	9
e	~~1111~~ ~~1111~~ ~~1111~~ 1	16
i	~~1111~~ ~~1111~~	10
o	~~1111~~ ~~1111~~	10
u	~~1111~~ 1	6

th~~e~~, h~~o~~rs~~e~~, st~~oo~~d, st~~i~~ll, t~~i~~ll, h~~e~~, h~~a~~d, f~~i~~n~~i~~sh~~e~~d, th~~e~~, hymn, wh~~i~~ch, J~~u~~d~~e~~, r~~e~~p~~ea~~t~~e~~d, ~~u~~nd~~e~~r, th~~e~~, sw~~a~~y, ~~o~~f, ~~a~~, p~~o~~lyth~~ei~~st~~i~~c, f~~a~~ncy, th~~a~~t, h~~e~~, w~~ou~~ld, n~~e~~v~~e~~r, h~~a~~v~~e~~, th~~ou~~ght, ~~o~~f, h~~u~~m~~ou~~r~~i~~ng, ~~i~~n, br~~oa~~d, d~~a~~yl~~i~~ght

Figure 1 Completed tally count

SAQ 2
Use the tally method to draw up a frequency table of the lengths of words for the data in Example 2.

Getting the computer to count
Having found a paper and pencil method of counting frequencies, we now need a method of getting the computer to do the counting of a list. The power of lists is derived from the apt use of the index. In Question 2 of Assignment 4 we saw how the items of two lists of data (name and telephone number-lists) were linked by a common index. The third member of the number-list was the telephone number for the third name in the name list, etc. Generally, the I-th member of the name list is linked to the I-th member of the number list.

Suppose we want to count the number of times the digits 0, 1, 2, ... 9 occur in a sequence. We can use ten counters:

C(0), C(1), C(2) ... C(9)

each of which will be zero at the start. To count the digits in 473808 we take the first digit in the sequence: 4. 1 is added to (4) and so on:

Digits entered	Counters after entry									
	C(0)	C(1)	C(2)	C(3)	C(4)	C(5)	C(6)	C(7)	C(8)	C(9)
start	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	1	0	0	1	0	0
3	0	0	0	1	1	0	0	1	0	0
8	0	0	0	1	1	0	0	1	1	0
0	1	0	0	1	1	0	0	1	1	0
8	1	0	0	1	1	0	0	1	2	0

So the idea is that when I is entered at the keyboard, increment C(I) by 1. This can usually be achieved in just two BASIC statements.

```
120 INPUT I
140 LET C(I) = C(I) + 1
```

A list counter

SAQ 3
The sequence

```
10 INPUT N
20 LET C(N) = C(N) + 1
```

is used to count the numbers of 0's, 1's, 2's etc. in the following input data: 3, 1, 0, 5, 9, 9, 6, 6, 6, 0, 4, 4, 2, 4, 1, 2, 1, 3, 0, 2, 1, 3. What are the values of the following:

(a) C(3) after three numbers have been input.
(b) C(9) after 12 numbers have been input.
(c) C(1) after all the numbers have been input.
(d) C(0) after all the numbers have been input.

We will now use this method of counting a list in an example.

Example 3
Write a program to input a sequence of single digits and to output the frequency with which each digit occurs.

Solution

A digit is one of the set of ten numbers 0, 1, 2, 3, ... 9. We will enter these one by one, with the sequence being terminated by −9999. So far, very routine!

The counting list will have ten counters:

C(0), C(1), C(2) ... C(9)

The program to solve the complete problem has two parts. (i) The input and increment routine incorporates the two statements 120 and 140 discussed above. (ii) The output routine is driven by a `FOR ... NEXT ...` loop, with index J running from 0 to 9.

```
 10 REM **COUNT THE NUMBER OF TIMES
 20 REM   EACH DIGIT IS ENTERED AND
 30 REM   STORE IN A COUNT-LIST C(I)**
 40 REM ******************************
 50 DIM C(10)
 60 CLS
100 PRINT "Input a list of single digits"
110 PRINT "end the list with -9999"
120 PRINT
130 REM **NEXT INPUT NOW**
140 INPUT ** "What is your next digit";I
150 IF I = -9999 THEN 190
160 C(I) = C(I) + 1
170 GOTO 130
190 REM ******************************
200 CLS
210 PRINT
220 PRINT "Digit", "Count"
230 PRINT
240 FOR J = 0 TO 9
250 PRINT J, C(J)
260 NEXT J
270 END
```

Input and counting routine

Printing table

Program 2 Counting with a list counter C(I) with , in line 220

```
 10 REM **COUNT THE NUMBER OF TIMES
 20       EACH DIGIT IS ENTERED AND
 30       STORE IN A COUNT-LIST C(I) **
 40 REM ******************************
 50 DIM C(10)
 60 CLS
100 PRINT "Input a list of single digits"
110 PRINT "end the list with -9999"
120 PRINT
130 REM **NEXT INPUT NOW**
140 INPUT "What is your next digit";I
150 IF I = -9999 THEN 190
160 C(I) = C(I) + 1
170 GOTO 130
190 REM ******************************
200 CLS
210 PRINT
220 PRINT "Digit      Count"
230 PRINT
240 FOR J = 0 TO 9
250 PRINT J,,C(J)
260 NEXT J
270 END
```

Program 2 Counting with a list counter C(I) (without , in line 220)

Typical output

(After entering 3, 7, 6, 4, 9, 1, 4, 9, 2, 7, 8, 0, 1, 5, 2, 7, −9999.)

```
RUN

Input a list of single digits
end the list with -9999

What is your next digit? 3
What is your next digit? 7
What is your next digit? 6
What is your next digit? 4
What is your next digit? 9
What is your next digit? 1
What is your next digit? 4
What is your next digit? 9
What is your next digit? 2
What is your next digit? 7
What is your next digit? 8
What is your next digit? 0
What is your next digit? 1
What is your next digit? 5
What is your next digit? 2
What is your next digit? 7
What is your next digit? -9999

Digit          Count
0              1
1              2
2              2
3              1
4              2
5              1
6              1
7              3
8              1
9              2

Ready
```

🄺 Program 2.

Frequency table for string lengths

We have written two programs so far in this unit; the first to find the lengths of strings, and the second to build up a frequency table.

In the following exercise we want you to combine these two ideas to build up a frequency table of lengths of words. If you wish you can use the words in the data already used in Example 2. Assume that the words will not be longer than 15 characters, so the length list will have elements:

L(1), L(2), L(3) ... L(15).

Exercise 1

Write a program to read in a set of words and to display a frequency table of their lengths.

5.4 Frequency diagrams

Frequency diagram for number of vowels
The picture of tally marks in Figure 1 makes a more immediate impact on us and somehow gives us more information about the distribution of frequencies of the vowels than just the column of figures. So why not get the computer to print a picture for us? You saw how to print rows of asterisks in Unit 4 by driving the print head across the page (or screen) with a `FOR ... NEXT` loop of variable range.

SAQ 4
What will appear on the screen as a result of the following program if you input 2, 5, 7, 8, 3 and 1?

```
10 INPUT A
20 FOR I = 1 TO A
30 PRINT "*";
40 NEXT I
45 PRINT
50 GOTO 10
```

Program 3

We can do the same thing using the frequencies from Figure 1 to determine the range and thus the number of asterisks printed across the page. This will generate a picture of the distribution.

Example 4
Write a program to print out a frequency diagram for the distribution of vowels given in Example 2.

Solution
Notice that this program draws the diagram from the frequencies we have already calculated. We input these frequencies in the lines 50 to 100.

We read the frequencies with a counter F(K) where F(1) is the number of a's, F(2) the number of e's, etc.

Then we print asterisks across the page according to the value of F(K) (lines 180 to 240).

```
 10 REM **FREQUENCY DISTRIBUTION**
 20 REM **PREPARATION PICTURE**
 30 REM **FREQUENCY-LIST IS F(K)**
 35 DIM F(6)
 40 K = 1
 44 CLS
 50 REM **INPUT ROUTINE**
 60 PRINT "Vowel",K
 70 INPUT "Enter frequency";F(K)
 80 IF F(K) = -9999 THEN 120
 90 K = K + 1
100 GOTO 50
110 REM ******************************
120 REM **DON'T ADD -9999 TO LIST**
130 N = K - 1
140 REM ******************************
150 REM **PRINT ROUTINE**
160 CLS
```

Reading the frequencies and storing them in F(1), F(2) ...

```
170 PRINT
180 FOR X = 1 TO N
190 FOR Y = 1 TO F(X)
200 PRINT "*";
210 NEXT Y                    Printing * across the page
220 PRINT
230 PRINT
240 NEXT X
250 REM ******************************
```

Program 4 Drawing a frequency distribution

```
RUN

Vowel          1
Enter frequency?  9
Vowel          2
Enter frequency? 16
Vowel          3
Enter frequency? 10
Vowel          4
Enter frequency? 10
Vowel          5
Enter frequency?  6
Vowel          6
Enter frequency? -9999

*********
****************
**********
**********
******
Ready
```

[K] Program 4.

Frequency diagram for length of words

If we want to draw a diagram of the frequencies with which the word lengths occurred in SAQ 1, we need to modify Program 4. Two modifications are necessary:

First the frequency list contains more items. There are 15 frequencies (1 to 15) plus −9999, so that's 16 items and we add `35 DIM F(16)` to Program 4.

Second the print routine at line 150 will run into problems when the frequency is zero. We can't drive the `FOR ... NEXT` loop from 1 to 0! So we must prevent the program going into the `FOR ... NEXT` loop when the frequency is zero. To do this we add `185 IF F(X) = 0 THEN 220`.

We must also modify the input instructions to:

```
60 PRINT "Length", K
70 INPUT "Enter frequency ";F(K)
```

So the program is

```
10 REM **FREQUENCY DISTRIBUTION**
20 REM **PREPARATION PICTURE**
30 REM **FREQUENCY-LIST IS F(K)**
35 DIM F(16)
40 K = 1
45 CLS
50 REM **INPUT ROUTINE**
```

```
 60 PRINT "Length", K
 70 INPUT "Enter frequency "; F(K)
 80 IF F(K) = -9999 THEN 120
 90 K = K + 1
100 GOTO 50
110 REM ******************************
120 REM **DON'T ADD -9999 TO LIST**
130 N = K - 1
140 REM ******************************
150 REM **PRINT ROUTINE**
160 CLS
170 PRINT
180 FOR X = 1 TO N
185 IF F(X) = 0 THEN 220
190 FOR Y = 1 TO F(X)
200 PRINT "*";
210 NEXT Y
220 PRINT
230 PRINT
240 NEXT X
250 REM ******************************
```

Program 4 (modified).

```
RUN

Length        1
Enter frequency? 1
Length        2
Enter frequency? 5
Length        3
Enter frequency? 4
Length        4
Enter frequency? 6
Length        5
Enter frequency? 9
Length        6
Enter frequency? 0
Length        7
Enter frequency? 1
Length        8
Enter frequency? 3
Length        9
Enter frequency? 1
Length       10
Enter frequency? 0
Length       11
Enter frequency? 0
Length       12
Enter frequency? 1
Length       13
Enter frequency? 0
Length       14
Enter frequency? 0
Length       15
Enter frequency? 0
Length       16
Enter frequency? -9999

*
*****
****
******
*********
```

```
*
***
*
Ready
```

[K] Program 4 (modified)

5.5 Tabulation

Note: We have tested the following programs using TAB on an Oric-1 computer and find that they do not work properly. There appears to be a 'bug' (i.e. a mistake) inside the Oric-1's BASIC ROM. We presume that corrected ROMs will be available for the Oric-1 in due course.

We've got the essential ingredients of a picture, but it is still far from being a meaningful diagram. It will help if we have the facility to move the print head across the page or screen to any predetermined position. In typing this is called tabulation (to arrange in tabular or table form). In BASIC the `TAB` function does this for us.

We take the same approach as we did in Unit 3, namely to write a snippet of program which explains itself – an approach well worth cultivating!

First, look at what happens if you number print positions across the screen:

```
50 PRINT "12345678901234567890123456789012"
60 PRINT "a";TAB (5);"e";TAB (7);"i";TAB (19);"o";TAB (31);"u"

RUN
12345678901234567890123456789012
a    e i           o           u
```

Program 5

You can see that `TAB (5)` printed `e` at the sixth position. Why? Because the machine counts print positions from position 0. This is demonstrated by Program 6 where the scale across the screen goes from 0:

```
50 PRINT "01234567890123456789012345678901"
60 PRINT "a";TAB (5);"e"; TAB (7);"i";TAB (19);"o";TAB (31);"u"

RUN
01234567890123456789012345678901
a    e i           o           u
```

Program 6

Now `TAB (5)` goes to the position labelled 5 but it is still the sixth position across the screen.

SAQ 5

Write a program to print `COL 1`, `COL 2`, `COL 3` across the screen with `COL 1` starting at position 0, `COL 2` at position 10 and `COL 3` at position 20.

Variable TAB and its effects

We can drive line 60 of the vowel print with a `FOR ... NEXT` loop to produce an actual table.

```
50 FOR I = 1 TO 7
60 PRINT "a";TAB (5);"e";TAB (7);"i";TAB (19);"o";TAB (31);"u"
70 NEXT I
```

Program 7

```
RUN
a    e i             o           u
a    e i             o           u
a    e i             o           u
a    e i             o           u
a    e i             o           u
a    e i             o           u
a    e i             o           u
```

Here is another example which shows how we can drive TAB with a variable. If we use TAB (V) where V is a variable, we can move the cursor to different positions across the screen. The program

```
30 FOR A = 1 TO 10
40 PRINT TAB (A);"Hello"
50 NEXT
60 STOP
```

Value of A in TAB (A) is determined by the loop variable A.

Program 8

produces

```
RUN
Hello
 Hello
  Hello
   Hello
    Hello
     Hello
      Hello
       Hello
        Hello
         Hello
```

We can go one step further and combine these two effects in one program:

```
50 FOR I = 0 TO 6
60 PRINT TAB (0+I);"a";TAB (5+I);"e";TAB (7+I);"i";TAB
  (19+I);"o";TAB (23+I);"u"
70 NEXT I
80 STOP
```

Program 9

```
RUN
a    e i             o           u
 a    e i             o           u
  a    e i             o           u
   a    e i             o           u
    a    e i             o           u
     a    e i             o           u
      a    e i             o           u
```

SAQ 6

Write a program segment to input three numbers of the user's choice which will place the string "heading" at three different positions across the same output line.

TAB and the frequency diagram

We are now in a position to set out the frequency diagram of Figure 2 in a more attractive manner.

The print routine of the modified Program 4 (lines 150 to 250) was:

```
150 REM **PRINT ROUTINE**
160 CLS
170 PRINT
180 FOR X = 1 TO N
185 IF F(X) = 0 THEN 220
190 FOR Y = 1 TO F(X)
200 PRINT "*";
210 NEXT Y
220 PRINT
230 PRINT
240 NEXT X
250 REM ******************************
```

Program 4 (modified)

We add:

line 172 to print column headings

line 174 to print a rule across the screen

line 176 to start the column divides (the rest of the divides we printed by the following loop)

line 182 (in the loop) prints X and F(X) across the page plus the column divides. This line ends in ";" which makes the next `PRINT` instruction (line 200) appear on the same line.

You will probably have to study this carefully to see all the detail in it:

```
 10 REM **FREQUENCY DISTRIBUTION**
 20 REM **PREPARATION PICTURE**
 30 REM **FREQUENCY-LIST IS F(K)**
 35 DIM F(16)
 40 K = 1
 45 CLS
 50 REM **INPUT ROUTINE**
 60 PRINT "Length",K
 70 INPUT "Enter frequency";F(K)
 80 IF F(K) = -9999 THEN 120
 90 K = K + 1
100 GOTO 50
110 REM ******************************
120 REM **DON'T ADD -9999 TO LIST**
130 N = K - 1
140 REM ******************************
150 REM **PRINT ROUTINE**
160 CLS
170 PRINT
172 PRINT "Length"; TAB (8);"Freq";TAB (18);"Tally"
174 PRINT "________________________________"
176 PRINT TAB (7);":";TAB (12);":"
180 FOR X = 1 TO N
182 PRINT TAB(2);X;TAB(7);":";TAB(9);F(X);TAB(12);":";
          TAB (14);
185 IF F(X) = 0 THEN 220
190 FOR Y = 1 TO F(X)
200 PRINT "*";
210 NEXT Y
220 PRINT
240 NEXT X
250 REM ******************************
```

Program 10 (with TAB)

```
RUN
Length    Freq       Tally  ---------------------------->172
-------------------------------- ---------------------->174
                     ---------------------------------->176
    1        1        *
    2        5        *****
    3        4        ****
    4        6        ******
    5        9        *********
    6        0
    7        1        *
    8        3        ***
    9        1        *
   10        0
   11        0
   12        1        *
   13        0
   14        0
   15        0
```

Effect of line 222 — as before but using TAB (14) as a base line (from line 182)

```
Ready
```

Ⓚ Program 10.

```
 10 REM**FREQUENCY DISTRIBUTION**
 20 REM **PREPARATION PICTURE**
 30 REM **FREQUENCY-LIST IS F(K)**
 35 DIM F(16)
 40 K = 1
 45 CLS
 50 REM **INPUT ROUTINE**
 60 PRINT "Length",K
 70 INPUT "Enter frequency";F(K)
 80 IF F(K) = -9999 THEN 120
 90 K = K + 1
100 GOTO 50
110 REM ******************************
120 REM **DON'T ADD -9999 TO LIST**
130 N = K - 1
140 REM ******************************
150 REM **PRINT ROUTINE**
160 CLS
165 LET Z = 0
170 PRINT
172 PRINT "Length   Freq    Tally"
174 PRINT "_______________________________"
176 PRINT TAB(20);":";TAB(17);":"
180 FOR X = 1 TO N
181 IF X > 9 THEN Z = -1
182 PRINT TAB(15);X;TAB(16+Z);":";TAB(14);F(X);":";TAB(14);
185 IF F(X) = 0 THEN 220
190 FOR Y = 1 TO F(X)
200 PRINT "*";
210 NEXT Y
220 PRINT
240 NEXT X
250 REM ******************************
```

Program 10: Lines 176 & 182 altered to suit TAB idiosyncrasy

Exercise 2
Modify Program 4 to give a print-out similar to that developed for Program 10 and to include the following points:
(a) an appropriate change of headings;
(b) a print-out of the letters a, e, i, o, and u as appropriate in the left-hand column;
(c) an appropriate scale at the base of the diagram.

5.6 Cutting up strings

Let's now look at a string which, though being an entity in its own right, contains more than one item of information. For example, 23 June 1971 is a single date but there are occasions when we only want to look at part of it, e.g. the month.

Filing dates
How many times have you been faced with a box on a form like this?

Date						
	D	D	M	M	Y	Y

If we look at D D M M Y Y the presentation has problems. Compare

23rd June 1971, or 230671

and 14th Sept 1973, and 140973.

The later date has the smaller number. Whereas with

4 July 1933, or 040733

and 15 Jan 1967, and 150167

the later date has the larger number. Clearly, then, D D M M Y Y is not very useful for filing dates.

The solution is to put the dates in the form Y Y M M D D. This makes the four dates above:

330704, 670115, 710623, and 730914

giving date and number consistency.

Dates are usually stored as numbers in the machine for use in calculations but are entered as strings to allow checking procedures to occur before they are stored.

If we are interested in a salary increment, then the year and month parts of the number would be important. If we are a music centre and send out reminders to our clients every three months to have their pianos tuned, then only the month may be important. The whole data string is important in its own right, but we can see that there may be valid reasons for cutting it up.

LEFT$(X$,I) and RIGHT$(X$,I)
If we want to consider part of a string, then we need a statement that will do this for us. We will start with two such statements.

LEFT$(X$,I) gives the left-most I characters of the string X$.

e.g. if X$ = "CUTTING"
then LEFT$(X$,3) = CUT

RIGHT$(X$,I) gives the right-most I characters of X$.

e.g. RIGHT (X$,4) = TING

Let's get the machine to tell us its own story. We enter a six-character string and use the index I of the FOR ... NEXT loop to peel off sub-strings of lengths 1 to 6. The scale (line 30) helps you to identify what's happening.

```
10 REM ** SLICING DEMONSTRATION **
20 CLS
30 INPUT "Enter a 6-character string"; X$
40 PRINT "          1234567890"
50 FOR I = I TO 6
60 LET A$ = LEFT$(X$,I)
70 PRINT I;"         ";A$
80 NEXT I
```

Program 11

[K] Program 11.

Left string run

```
RUN

Enter a 6-character string? 123456
              1234567890
1             1
2             12
3             123
4             1234
5             12345
6             123456

Ready
```

Now if we change line 60 of Program 11 to

```
60 LET A$ = RIGHT$(X$,I)
```

and enter the string abcdef the result is:

Right string run

```
RUN

Enter a 6-character string? abcdef
              1234567890
1             f
2             ef
3             def
4             cdef
5             bcdef
6             abcdef

Ready
```

[K] Program 11. Then change line 60 for RIGHT$

SAQ 7
If A$ = 1A2B3C4D, what are the following:

(a) LEFT$(A$,1) (c) RIGHT$(A$,3)
(b) LEFT$(A$,4) (d) RIGHT$(A$,4)

Cutting up strings of variable length
Program 11 is a bit awkward because we had to specify (in line 50) how long the string was to be: six characters. But we might want to input strings of any length. This is easily done by modifyng Program 11 so that the computer measures the length of the string we input and runs that length to control the FOR ... NEXT loop. The modifications are:

```
10 REM ** STRING TEST **
20 CLS
30 INPUT "Enter a string";X$
40 PRINT "          1234567890"
50 FOR I = I TO LEN(X$) ]         LEN(x$) acts as the upper
60 LET A$ = LEFT$(XS,I) ]         limit of the loop
70 PRINT I;"          ";A$
80 NEXT I
```

Program 12

```
Enter a 6-character hamstring
              123456789
1             h
2             ha
3             ham
4             hams
5             hamst
6             hamstr
7             hamstri
8             hamstrin
9             hamstring
```

[K] Program 12.

Exercise 3
Write a program which allows you to input words one at a time and which will output those words which begin with a vowel.

Exercise 4
Write a program which will change the output of Program 12 to:

```
     f
    ef
   def
  cdef
 bcdef
abcdef
```

MID$(X$,I,J)
We have used LEFT$ and RIGHT$ to cut sections off either end of a string, but we might want a section in the middle of a string, e.g. `M M` in `Y Y M M D D`. There is

another BASIC statement that will give us a section of this type:

```
MID$(X$,I,J)
```

This will cut a sub-string of length J, starting from position I:

```
MID$(X$,I,J)
```

length of sub-string from position I

position in the string

e.g. if X$ = position

```
MID$(X$,5,2) = ti
```

and MID$(X$,2,4) = osit

We will use the computer again to demonstrate MID$ at work by a further modification to Program 11. We have already adapted Program 11 to allow us to input a string of any length.

If we now change line 60 to

```
60 LET A$ = MID$(X$,I,1)
```

we get

```
10 REM ** STRING TEST **
20 CLS
30 INPUT "Enter a string";X$
40 PRINT"           1234567890"
50 FOR I = 1 TO LEN(X$)
60 LET A$ = MID$(X$,I,1)
70 PRINT I;"          ";A$
80 NEXT I
```

Program 13

If we input shoestring we get:

```
RUN

Enter a string? shoestring
         1234567890
1        s
2        h
3        o
4        e
5        s
6        t
7        r
8        i
9        n
10       g

Ready
```

Here MID$ is looking at all possible sub-strings of length 1.

If we now use

```
60 LET A$ = MID$(X$,1,2)
```

and input stringent we get:

```
RUN

Enter a string? stringent
         1234567890
1        st
2        tr
```

```
3          ri
4          in
5          ng
6          ge
7          en
8          nt
9          t

Ready
```

'null' string

[K] Program 13.

The last sub-string caused problems. We can't get a sub-string two characters long from a string of nine characters starting at the ninth character. In trying to do so we enter a default state, and are given a null-string as a reward. There must always be enough characters left of the original string to take out the sub-string.

Generally, if we wish to take out J characters we will not be able to start this sub-string beyond the (LEN(X$) – J + 1)th position.

Yes, + 1.

e.g., if LEN(X$) = 10 and J = 3, then LEN(X$) – J = 7;

but we can get a string of length 3 from a string of length 10 if we start at 8, i.e. character positions 8, 9, 10 of the original string.

SAQ 8

Write a program to accept as input London telephone numbers in the form 01 XXX XXXX and output the exchange codes only. (Remember that the spaces are characters just as much as the digits.)

Mid-string program

As you have probably spotted, `MID$` can cut left sub-strings and right sub-strings if we want *it* to. In other words, it can give us every possible sub-string. Here is a program that does this. First it prints out all sub-strings of length 1, then all of length 2 and so on until it prints the whole word which is the only sub-string of the same length as the word itself!

```
10 REM ** STRING TEST **
20 CLS
30 INPUT "Enter a string ";X$
40 PRINT "J      I   1234567890"
45 FOR J = 1 TO LEN(X$)
50 FOR I = 1 TO (LEN(X$) - J + 1)
60 LET A$ = MID$(X$,I,J)
70 PRINT J;"     ";I;" ";A$
80 NEXT I
85 PRINT " "
90 NEXT J
100 END

RUN
```

J is the length of the sub-string, starting at I.

Program 14

```
RUN
Enter a string? string
j         I    1234567890
1         1    s
1         2    t
1         3    r
1         4    i
1         5    n
1         6    g


2         1    st
2         2    tr
2         3    ri
2         4    in
2         5    ng


3         1    str
3         2    tri
3         3    rin
3         4    ing


4         1    stri
4         2    trin
4         3    ring


5         1    strin
5         2    tring


6         1    string

Ready
```

Note headings and scale

🄺 Program 14.

5.7 VAL

Having found a method of cutting up strings, we now need a method of examining what we have got. One such method is to use `VAL(A$)` which looks at the numeric value of A$.

`VAL(A$)` on the Oric gives us the numerical value of the string A$.

Program to demonstrate VAL
The following program gives `VAL(N$)` for any string you input.

```
 10 REM ** THE VAL FUNCTION **
 15 CLS
 20 INPUT "Next string";N$
 25 IF N$="zzzz" THEN 999
 30 N = VAL (N$)
 40 P = VAL(LEFT$(N$,2))
 50 Q = VAL(MID$(N$,3,2))
 60 PRINT " "
 70 PRINT N,P,Q
 80 PRINT " "
 90 GOTO 10
999 END
```

Program 15

```
RUN
NEXT STRING?    123456
123456          12       34

NEXT STRING?    12345A
12345           12       34

NEXT STRING?    1234AB
1234            12       34

NEXT STRING?    123ABC
123             12        3

NEXT STRING?    12ABCD
12              12        0

NEXT STRING?    1ABCDE
1               1         0

NEXT STRING?    ABCDEF
0               0         0
```

VAL still gives the value of the left-most digits, even though the string contains AB-non-numeric characters.

A is now at position 3, so VAL(A) = 0

🄺 Program 15

SAQ 9
What are the values of the following?

(a) `VAL(A$)` where A$=54
(b) `VAL(B$)` where B$=76xy
(c) `VAL(C$)` where C$=A3
(d) `VAL(D$)` where D$=-132
(e) `VAL(LEFT$(E$,2))` where E$=593
(f) `VAL(LEFT$(F$,1))` where F$=8am
(g) `VAL(LEFT$(G$,2))` where G$=Z35
(h) `VAL(RIGHT$(H$,1))` where H$=593
(i) `VAL(RIGHT$(I$,2))` where I$=8am
(j) `VAL(RIGHT$(J$,3))` where J$=z35

Date string check
We are now in a position to see how `VAL` can be put to practical use; in this case to check the accuracy of dates keyed into a computer. This is typical of what happens in computers all the time. We know errors will frequently happen when data is keyed in so, wherever possible, we try to use the computer to detect the errors.

Example 5
Write a program to carry out data checks on the three fields of a six-digit date-string.

Solution
The date-string D$ has three fields in the form:

```
YY          MM            DD
 \            \             \RIGHT$(D$,2)
  \          MID$(D$,3,2)
  LEFT$(D$,2)
```

We are going to consider the years 1980 and 1981 only, so:

`VAL(LEFT$(D$,2))` should have a range of 80-81,
`VAL(MID$(D$3,2))` should have a range of 1-12,
`VAL(RIGHT$(D$,2))` should have a range of 1-31.

This is a fairly complex process as we first need to decide on the steps involved. These are given in the following flowchart (Figure 2).

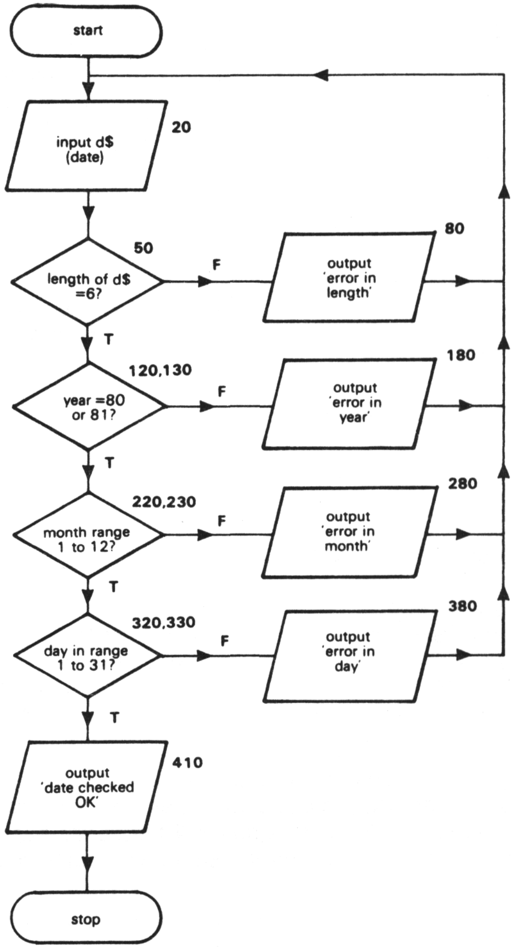


Figure 2 Flowchart showing the four checks on YYMMDD

The program is fairly straightforward as it goes through each of the four checks. If any check fails, the program prints an error message and returns control to line 20. If there are no errors it runs through to line 410 where the correct entry is confirmed.

```
 10 REM ** DATE CHECK **
 15 CLS
 20 INPUT "Next date ";D$
 30 IF D$="ZZZZ" THEN 900
 50 IF LEN (D$) = 6 THEN 110          Length check
 80 PRINT"!!!!!!Error in date length!!!!!!"
 90 GOTO 20                           Included here for demon-
100 REM ***********************       stration purposes; would
110 PRINT "String length correct"     not normally appear in a
120 IF VAL (LEFT$(D$,2)) = 80 THEN 210  checking routine
130 IF VAL (LEFT$(D$,2)) = 81 THEN 210
180 PRINT"!!!!!!Error in year field!!!!!!"  Year check
190 GOTO 20
200 REM ***********************
210 PRINT "Year field correct"
220 IF VAL (MID$(D$,3,2)) < 1 THEN 280
230 IF VAL (MID$(D$,3,2)) < = 12 THEN 310   Month check
280 PRINT"!!!!!!Error in month field!!!!!!"
290 GOTO 20
300 REM ***********************
310 PRINT "Month field correct"
320 IF VAL (RIGHT$(D$,2)) < 1 THEN 380
330 IF VAL (RIGHT$(D$,2)) < = 31 THEN 410   Day check
380 PRINT"!!!!!!Error in day field!!!!!!"
390 GOTO 20
400 REM ***********************
410 PRINT "Date string within check limits"
490 GOTO 20
900 PRINT "End of date check"
```

Program 16

Typical run

```
RUN
Next date? 1234567
!!!!!!Error in date length!!!!

Next date? 123456
String length correct
!!!!!!Error in year field!!!!

Next date? 803456
String length correct
Year field correct
!!!!!!Error in month field!!!!

Next date? 801256
String length correct
Year field correct
Month field correct
!!!!!!Error in day field!!!!

Next date? 800131
String length correct
Year field correct
Month field correct
Date string within check limits
ZZZZ
End of date check

Ready
```

Ⓚ Program 16.

5.8 Printing: PLOT

So far we have displayed information on the screen in a serial manner: we have printed each line as it comes, spaced out, as appropriate, with blank lines (generated by PRINT).

We can print at any character position on the screen by using PLOT.

This has the format

```
PLOT x,y,A$
```

where x is the character position across the screen, ranging from 1 at the left-hand side of the screen, to 38 at the right-hand side. You will be familiar with this from the TAB statement. Y is the line number, remembering that there are 27 lines on the screen upon which one may print, and that numbers of lines increase downwards, from 0 at the top to 26 at the bottom.

TAB allows the computer to display in the way that a typewriter does; to reach any point further down a typed sheet, you have to move the carriage down and the print head across. PLOT allows you to 'zoom in' on any character position on the screen without having to use blank PRINT statements or TABs to move there.

If you want, for example, to output a grid of stars you could use either of the following programs, 17a or 17b to achieve the desired effect.

```
10 REM ** PRINT **
15 CLS
17 PRINT
20 FOR J = 1 TO 21 STEP 3
30 FOR I = 1 TO 37 STEP 2
40 PRINT "* ";
50 NEXT I
60 PRINT
70 PRINT
80 NEXT J
```

Program 17a Star grid: PRINT

```
10 REM ** PLOT **
15 CLS
17 PRINT
20 FOR J = 1 TO 21 STEP 3
30 FOR I = 1 TO 37 STEP 2
40 PLOT I,J,"*"
50 NEXT I
60 PRINT
70 PRINT
75 PRINT
80 NEXT J
```

Program 17b Star grid:PLOT

```
RUN
*******************
*******************
*******************
*******************
*******************
*******************
*******************
Ready
```

Assignment 5

1. Write a program to find the frequency with which each vowel occurs in the words in the Jude data of Example 2, giving also a summary of the total number of vowels and consonants which occur in these words.
2. Write a program to input a string of characters and to output this string in reverse order.

Objectives of Unit 5

Check that you are now able to write simple programs:

Using `LEN(A$)` □
Using `LET C(I) = C(I) + 1` to count frequencies □
To print a frequency diagram □
Using `TAB` to print in columns □
Using `TAB` to print a frequency table with headings and scale □
Using `LEFT$(X$,I)` and RIGHT$(X$,I) □
Using `MID$(X$,I,J)` □
Using `VAL(A$)` □
Using PLOT x,y,A$ □

Answers to SAQs and Exercises

SAQ 1
(a) 3; (b) 1; (c) 2 (not 72 – `LEN` counts the number of characters);
(d) 7 (`LEN` counts the characters regardless of whether they are numbers, letters or spaces).

SAQ 2
Your answer should be:

Word length	Count	Total
1	1	1
2	~~1111~~	5
3	1111	4
4	~~1111~~ 1	6
5	~~1111~~ 1111	9
6		0
7	1	1
8	111	3
9	1	1
10		0
11		0
12	1	1
13		0
14		0
15		0

SAQ 3

(a) C(3) = 1 (Not 3 or 4! C(3) has counted the number of 3's input.)
(b) C(9) = 2
(c) C(1) = 4
(d) C(0) = 3

Exercise 1

The solution appears in the following text.

SAQ 4

```
**
*****
*******
********
***
*
```

SAQ 5

```
10 PRINT "COL 1";TAB(9);"COL 2";TAB(19);"COL 3"
```

SAQ 6

```
10 INPUT A
20 INPUT B
30 INPUT C
40 PRINT TAB(A);"heading";TAB(B);"heading";TAB(C);"heading"
```

Program 18

Exercise 2

The program asks you to enter the vowels one by one, and their frequencies.

```
 10 REM ** FREQUENCY DISTRIBUTION **
 20 REM ** PREPARATION PICTURE **
 30 REM ** FREQUENCY-LIST IS F(K) **
 35 DIM F(5)
 40 DIM V$(5)
 45 CLS
 50 REM ** INPUT ROUTINE **
 55 FOR K = 1 TO 5
 60 INPUT "Vowel     ";V$(K)
 65 INPUT "Frequency ";F(K)
 70 PRINT
100 NEXT K
110 *****************************
150 REM ** PRINT ROUTINE **
160 CLS
170 PRINT
180 PRINT "Vowel    Freq    Tally"
190 PRINT ========================"
200 FOR X = 1 TO 5
210 PRINT " ";V$(X);"    :  ";F(X);
212 IF F(X) > 9 THEN 214
213 PRINT " ";
214 PRINT "  :  ";
215 IF F(X) = 0 THEN 250
220 FOR Y = 1 TO F(X)   <--- print out from word list
230 PRINT "*";
240 NEXT Y
250 PRINT
260 PRINT
270 NEXT X
275 PRINT "......Scale......0......5......0......5......0"
280 REM *************************
```

Program 19

```
RUN
Vowel     ?  a
Frequency?  9
Vowel     ?  e
Frequency?  16
Vowel     ?  i
Frequency?  10
Vowel     ?  o
Frequency?  10
Vowel     ?  u
Frequency?  6

Vowel    Freq       Tally
===============================
  a        9     *********
  e        16    ****************
  i        10    **********
  o        10    **********
  u        6     ******
....Scale.... 0....5....0....5.

Ready
```

Ⓚ Program 19

SAQ 7

(a) 1;b3 (b) 1A2B; (c) 4D; (d) 3C4D.

Notice that LEFT$ and RIGHT$ treat all characters in a string the same way. It doesn't matter whether they are numbers or letters, they still get counted.

Exercise 3

```
   5 CLS
  10 REM ** IS LEFT-MOST CHARACTER A VOWEL? **
  20 INPUT "Next word ";W$
  30 IF W$ = "ZZZZ" THEN 9999
  40 L$ = LEFT$(W$,1)
  50 IF L$ = "a" THEN 200
  60 IF L$ = "e" THEN 200
  70 IF L$ = "i" THEN 200
  80 IF L$ = "o" THEN 200
  90 IF L$ = "u" THEN 200
 100 GOTO 10
 105 PRINT
 190 REM ***********************
 200 PRINT
 210 PRINT $,,W$
 215 PRINT
 220 GOTO 10
 230 REM ***********************
9999 END
```

Program 20

```
RUN
Next word?        albatross
a                 albatross
Next word?        unicorn
u                 unicorn
Next word?        eagle
e                 eagle
Next word?        insect
i                 insect
```

```
Next word?          ostrich
o                   ostrich
Next word?          zebra
z                   zzzz

Ready
```

Ⓚ Program 20

Exercise 4

```
 10 REM ** SLICING VARIABLE STRINGS **
 15 CLS
 20 INPUT "Enter 6-character string";X$
 30 PRINT "             1234567890"
 40 FOR I = 1 TO 6
 50 A$ = RIGHT$(X$,I)
 60 PRINT I;
 70 FOR Z = 1 TO 14-I
 80 PRINT " ";
 85 NEXT Z
 90 PRINT A$
100 NEXT I
110 END
```

Program 21

```
RUN
Enter 6-character string?  abcdef
               1234567890
1                   f
2                  ef
3                 def
4                cdef
5               bcdef
6              abcdef
```

Ⓚ Program 21.

SAQ 8

```
10 REM ** LONDON PHONE NUMBERS **
20 CLS
30 INPUT "Next telephone number ";N$
40 PRINT N$
50 A$ = MID$(N$,4,3)
60 PRINT A$
70 GOTO 30

RUN
Next telephone number? 01-580-4411
01-580-4411
580

Ready
```

Program 22

Ⓚ Program 22.

SAQ 9

(a) 54; (b) 76 (stops at letters); (c) 0 (starts with a letter. Therefore 0); (d) −132; (e) 59; (f) 8; (g) 0 (starts with a letter); (h) 3; (i) 0 (starts at a which is a letter); (j) 35.

UNIT 6
Mainly about dice and games

6.1 Random numbers

The programming function which allows us to inject a sense of fun into a program is the one which generates random numbers. This function is at the heart of many of the game-playing and simulation programs which are now available for microcomputers.

You will have met random numbers when playing games; games which involve tossing a coin or throwing a die, or drawing numbers out of a hat. These domestic games have become institutionalised in casinos, bingo clubs, the ritualistic draw for the FA Cup competition and, of course, on a larger scale, the monthly draw for premium bonds. Although we all have an intuitive idea of what we mean by a sequence of random numbers, it is quite difficult to define the idea clearly. Let's have a look at some number sequences to try and clarify this idea.

Here are three 'thought experiments', each of which involves throwing a six-sided dice 15 times. Imagine that in the first experiment the uppermost values of the dice had those values shown in sequence A shown in Figure 1. The second experiment generated the numbers shown in sequence B and the third experiment gave us the numbers shown in sequence C.

Sequence A
5,1,2,4,6,3,2,1,6,3,5,4,3,4,2

Sequence B
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6

Sequence C
1,2,3,4,5,6,1,2,3,4,5,6,1,2,3

Figure 1 Random sequences?

Most of us would be quite happy that sequence A represented a typical sequence of numbers generated by throwing a die 15 times. This number sequence shows no definable patterns or repetitions and each number occurrence would seem to be 'equally likely'. We are not surprised at the appearance of any of the sub-sequences in this main sequence. By contrast, however, sequence B is quite unreasonable. We would certainly not expect to have thrown 15 sixes with 15 consecutive throws of the dice. We would be highly suspicious had this happened and we would blame a weighted dice. Intuitively, we would be prepared to accept that sequence A had occurred 'by chance' but would not be prepared to accept that this was so for sequence B.

Another feature about random number sequences which we learn by experience is that, in long sequences, localised 'unfair' occurrences iron themselves out. What we mean by this is that after, say, a hundred throws, we would expect on average about 16 ones, about 16 twos, 16 threes and so on. In other words, over a longer sequence we expect the 'laws of chance' to apply. If we now consider sequence C with its emerging pattern '... 6,1,2,3,4,5,6,1 ...' continued for a hundred throws then this long term averaging-out effect would be satisfied. But once again this sequence would not be intuitively acceptable to us as random because we would not expect this sequential pattern to persist over a hundred throws by chance alone.

These concepts of 'statistical averaging' over a sequence of throws and of the 'reasonableness' of the patterning of the numbers in sequence are intuitively acquired from games of chance. There are statistical techniques to test these two

feaures of a random number sequence, but we will not be concerned with those techniques here.

A computer is a very determinate machine. You will, therefore, not be surprised to learn that quite special features have to be programmed into the machine to achieve a sequence of random numbers. For our uses, however, we will assume that a table of random numbers has been stored in the machine's memory. The sequence of numbers is very long and generation would have to occur for a long time before the sequence repetition became apparent. To achieve a different random number sequence from one program execution to another, all that a machine has to do is to start reading this table of random numbers from a different point. This starting point is often referred to as the 'seed' and we talk about random number sequences as starting from different seeds. Because the computer has to 'contrive' random number sequences, the numbers produced are usually referred to as pseudo-random numbers.

6.2 The RND function

If you run the following program you will be able to see the effect of RND.

```
10 REM **Generate ten random numbers**
15 CLS
20 FOR I = 1 TO 10
30 B = RND(1)
40 PRINT B
50 NEXT I
```

Program 1 RND

A typical run produces numbers like:

```
RUN
 .253772171
 .128916266
 .636510833
 .956083769
 .497292619
 .481585843
 .73278014
 .675114126
 .18262696
 .782880158
Ready
```

[K] Program 1.

These certainly look like random numbers and, if you key RUN, you will get a completely different list. Actually the computer has a list of jumbled up numbers and RND plunges into that list and reads as many numbers as you ask for.

The statement B = RND(1) makes the variable B equal to a random number in the range 0 to 1 (but always less than 1).
If you use a value X, such that

```
A = RND(X)
```

then if X is greater than or equal to 1, the random number will be in the range 0 to 1.
If X equals zero, the random number will be the one most recently generated, for example:

```
10 REM **Generate ten random numbers**
15 CLS
20 FOR I = 1 TO 10
30 B = RND(0)
40 PRINT B
50 NEXT I
```

Program 2a

```
RUN

 .782880158
 .782880158
 .782880158
 .782880158
 .782880158
 .782880158
 .782880158
 .782880158
 .782880158
 .782880158

Ready
```

If X is less than zero (as in Program 2b below), the number produced is the same, whatever value X has.

```
10 REM **Generate ten random numbers**
15 CLS
20 FOR I = 1 TO 10
30 B = RND(-2)
40 PRINT B
50 NEXT I
```

Program 2b

RND (1)

What the above investigation demonstrates is that

`RND(1)` will give random numbers within the range 0-1.

How can we extend the range to get other random numbers? Quite simply by multiplying `RND(1)` by another number. So:

`RND(1)` gives a random number in the range 0-1;
`6*RND(1)` gives a random number in the range 0-6; and
`52*RND(1)` gives a random number in the range 0-52;
etc.

You can think of `RND(1)` as a 'conversion factor' which changes at will. The following program tries out this idea.

```
 10 REM **RND as a conversion factor**
 20 PRINT " I         RND(1)         RND(1) * 6"
 30 PRINT "---        -----          ----------"
 40 REM **Number generation loop**
 50 FOR I=1 TO 10
 60 B = RND(1)
 70 C = 6 * B
 80 PRINT I;TAB (8);B;TAB (20);C
 90 NEXT I
100 END
```

Program 3 (with TAB)

```
 10 REM **RND as a conversion factor**
 15 CLS
 20 PRINT " I        RND(1)        RND(1) * 6"
 30 PRINT "---      -----         ----------"
 40 REM **Number generation loop**
 50 FOR I = 1 TO 10
 60 B = RND(1)
 70 C = 6 * B
 80 PRINT " ";I;
 82 IF I < 10 THEN PRINT " ";
 84 PRINT "   ";B;
 85 LET B$ = STR$(B)
 86 FOR X = 1 TO (13 - LEN(B$))
 88 PRINT " ";
 90 NEXT X
 92 PRINT C
 94 NEXT I
100 END
```

Program 3 (without TAB)

```
RUN
 I        RND(1)          RND(1) * 6
---       ---------       ----------
 1        .252222631      1.51333579
 2        .812027557      4.87216534
 3        .0398211227     .238926736
 4        .999940628      5.99964377
 5        .0908466404     .545079843
 6        .0497669884     .29860193
 7        .970071052      5.82042631
 8        .0736029557     .441617734
 9        .646363964      3.87818378
10        .407537336      2.44522402
Ready
```

Run with line 70 as

`70 C = 52 * B`

plus change of headings in line 20:

```
 I        RND(1)          RND(1) * 52
---       -----           -----------
 1        .242709787      12.6209089
 2        .691334289      35.949383
 3        .525909661      27.3473024
 4        .447893187      23.2904457
 5        .802253701      41.7171925
 6        .0758692365     3.9452003
 7        .855632893      44.4929105
 8        .656634486      34.1449933
 9        .703331943      36.5732611
10        .921806395      47.9339325
Ready
```

[K] Program 3.

SAQ 1

Write a program to print out six random numbers in the range 0 to 5.999999.

The RND(1) + 1 function

If you look again at the output of Program 3 on the run with `6 * RND(1)`, the numbers were:

```
1.51333579
4.87216534
.238926736
5.99964377
.545079843
.29860193
5.82042631
.441617734
3.87818378
2.44522402
```

Look now at the numbers before the decimal point. They are:

1,4,0,5,0,0,5,0,3,2

i.e. members of the set

(0,1,2,3,4,5)

But, if we were throwing dice, we would generate members of the set (1,2,3,4,5,6). All we have to do, then, is to add 1 to each member of the first set to get the second.

Now in games we frequently want to throw a die (outcomes 1,2,3,4,5,6) or use a pack of cards (52 outcomes) so we are particularly interested in the functions

```
6 * RND(1) + 1  and  52 * RND(1) + 1
```

The following program allows us to explore these.

```
 10 REM **RND(1) + 1 as a conversion factor**
 15 CLS
 20 PRINT " I            RND(1)          RND(1) * 6 + 1"
 30 PRINT "---          -----           --------------"
 40 REM **Number generation loop**
 50 FOR I = 1 TO 10
 60 B = RND(1)
 70 C = 6 * B + 1
 80 PRINT " ";I;
 82 IF I < 10 THEN PRINT " ";
 84 PRINT "   ";B;
 85 LET B$ = STR$(B)
 86 FOR X = 1 TO (13 - LEN(B$))
 88 PRINT " ";
 90 NEXT X
 92 PRINT C
 94 NEXT I
100 END
```

*Program 4 RND(1) * 6 + 1 (without TAB)*

```
 10 REM **RND(1) + 1 as a conversion factor**
 15 CLS
 20 PRINT " I        RND(1)          RND(1) * 6 + 1"
 30 PRINT "---      -----           --------------"
 40 REM **Number generation loop**
 50 FOR I = 1 TO 10
 60 B = RND(1)
 70 C = 6 * B + 1
 80 REM **Note the + 1**
 90 PRINT I;TAB(8);B;TAB(20);C
100 NEXT I
110 END
```

*Program 4 RND(1) * 6 + 1 (with TAB)*

```
RUN
 I          RND(1)          RND(1) * 6 + 1
---         -----           --------------
 1          .844115356      6.06469214
 2          .54301045       4.2580627
 3          .667422996      5.00453798
 4          .913922605      6.48353563
 5          .994783023      6.96869814
 6          .0832884136     1.49973048
 7          .193592664      2.16155599
 8          .153782162      1.92269297
 9          .636740856      4.82044513
10          .0931283177     1.55876991
Ready
```

[K] Program 4.

The INT function

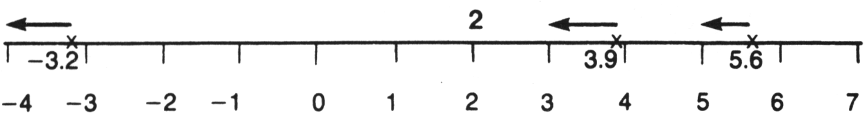
If you now look at the numbers before the decimal point in the run of Program 4, you will see that we have generated the random numbers we needed. Column 3 has the numbers from 1 to 6; if you change line 70 to L E T C = 52 *B + 1, alter the column title in line 20, and run Program 4 again, you will find column 3 has numbers from 1 to 52.

But what about all the garbage to the right of the decimal point? Well, we have a function to get rid of that: the `INT` function.

The effect of `INT(X)` is to give the whole number (or integer part) of the number `X`, i.e. the largest integer which is not larger than `X`. The effect of `INT` is to 'chop' down to the next highest whole number:

```
INT(5.6) = 5
INT(3.9) = 3
INT(-3.2) = -4
INT(2) = 2
```

If `INT(-3.2) = -4` surprises you, look at the number line and remember that `INT` always chops down to the next whole number. It doesn't 'round' numbers.



SAQ 2

What are the values of the following:

(a) `INT(4.5)`
(b) `INT(9.1)`
(c) `INT(-2.5)`
(d) `INT(-0.99)`
(e) `INT(1.01)`

Now, with `INT`, we can at last generate the whole numbers from 1 to 6 and from 1 to 52 to use as dice or cards. All we need is

`INT(6 * RND(1) + 1)`
and `INT(52 * RND(1) + 1)`

The following program prints out the values of these two functions:

```
 10 REM **The INT function**
 15 CLS
 20 PRINT " I";;TAB (5);"RND(1)";TAB (12);"INT(6*---";
    TAB (22);"INT(52*--"
 30 PRINT TAB (12);"---RND(1) + 1)";TAB (22);"---RND(1) + 1)"
 40 PRINT "---";TAB (5);"-----";TAB (12);"---------";
    TAB (22);"---------"
 50 PRINT
 60 REM **Start of loop**
 70 FOR I = 1 TO 10
 80 B = RND(1)
 90 C = INT(6 * B + 1)
100 D = INT(52 * B + 1)
110 PRINT I;TAB (4);B;TAB (18);C;TAB (25);D
120 NEXT I
130 END
```

Program 5 (with TAB) INT to give whole numbers

```
 10 REM **The INT function**
 15 CLS
 20 PRINT " I   RND(1) INT(6*---      INT(52*--"
 25 PRINT "            -RND(1) + 1 -RND(1) + 1"
 30 PRINT "---  -----  ---------   -----------"
 50 PRINT
 60 REM **Start of loop**
 70 FOR I = 1 TO 10
 80 B = RND(1)
 90 C = INT(6 * B + 1)
100 D = INT(52 * B + 1)
110 PRINT I;
112 IF I < 10 THEN PRINT " ";
114 PRINT "  ";
116 PRINT B;
118 B$ = STR$(B)
119 FOR X = 1 TO (13 - LEN(B$))
120 PRINT " ";
121 NEXT X
122 PRINT C;"    ";D
124 NEXT I
130 END
```

Program 5 (without TAB)

```
z
RUN
 I         RND(1)          INT(6*---     INT(52*--
                           -RND(1) + 1   -RND(1) + 1
---        -----           ---------     -----------
 1         .522031701          4             28
 2         .637713894          4             34
 3         .590600008          4             31
 4         .406692078          3             22
 5         .825937902          5             43
 6         .631701922          4             33
 7         .58208614           4             31
 8         .617291381          4             33
 9         .20620989           2             11
10         .8496742            6             45
Ready
```

🄺 Program 5.

6.3 Random number postscript

The following program simulates the tossing of a die a hundred times:

```
 10 REM **100 tosses of a die**
 15 CLS
 20 REM **J loop to form each row**
 30 REM **I loop to determine number of rows**
 40 FOR I = 1 TO 10
 50 FOR J = 1 TO 10
 60 X = INT(6 * RND(1) + 1)
 70 PRINT X;" ";
 80 NEXT J
 90 PRINT
100 PRINT
110 NEXT I
120 END
```

Program 6 Die toss

```
RUN
4 6 6 1 3 3 6 6 5 1

1 2 4 2 6 2 1 2 6 2

5 5 2 4 1 3 2 1 5 4

4 3 3 3 6 6 6 4 2 5

2 6 1 3 6 5 1 6 1 5

6 6 5 3 3 4 4 3 4 1

1 3 6 2 6 5 4 6 3 3

6 1 2 4 1 2 6 4 4 1

6 5 6 6 5 4 3 1 1 2

6 3 2 2 5 4 6 2 2 2
Ready
```

[K] Program 6.

Just to make sure that you understand the `INT` function, try the following questions.

SAQ 3

The program:

```
10 REM **SAQ 3**
15 CLS
20 FOR X = -3.8 TO -1.8 STEP .2
30 Y = INT(X)
40 PRINT X,Y
50 NEXT X
60 END
```

Program 7

prints out ten pairs of numbers. What are they?

SAQ 4
The program:

```
10 REM **SAQ 4**
15 CLS
20 FOR X = 1.6 TO 3.4 STEP .2
30 Y = INT(X)
40 PRINT X,Y
50 NEXT X
60 END
```

Program 8

prints out nine pairs of numbers. What are they?

6.4 Two examples

This section is made up of two lengthy examples. We suggest that you try to treat them as exercises first and then compare your solution with ours.

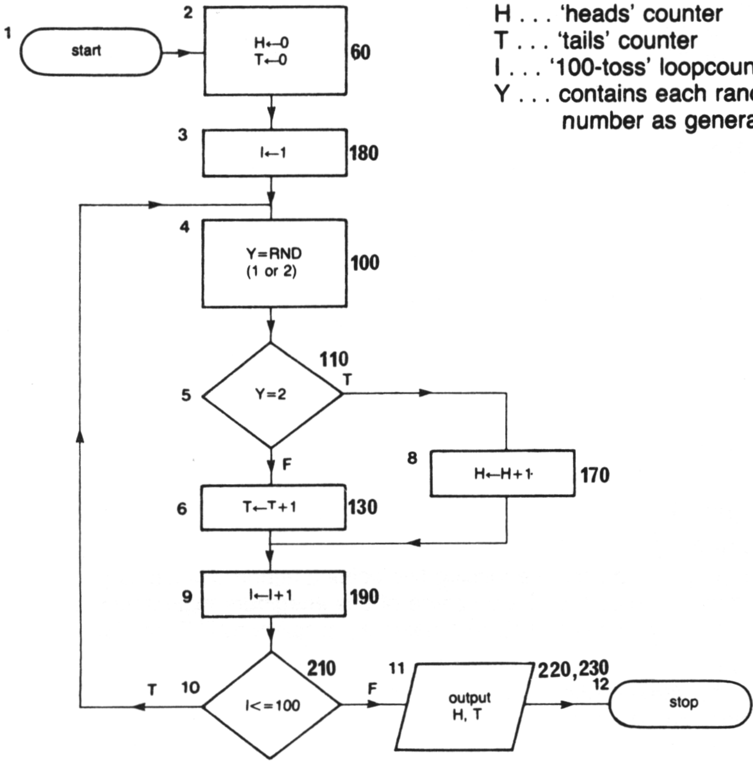
Example 1
Write a program to simulate tossing a coin a hundred times. Count and output the number of times the coin falls heads and tails.

Solution
The heart of the solution is a random number generator which produces a 1 or a 2. We will use 1 to represent a tail and 2 to represent a head. Using this approach, a descriptive algorithm for the solution to the problem is:

1. Start.
2. Set heads total and tails total to zero.
3. Start loop counter.
4. Generate the two values 1 and 2 randomly.
5. If random number equals 2 then go to statement 8 otherwise go on to statement 6.
6. Add 1 to tails total.
7. Go on to statement 9.
8. Add 1 to heads total.
9. Add 1 to loop counter.
10. If loop counter <=100 then go to statement 4 otherwise go on to statement 11.
11. Output total heads and total tails.
12. Stop.

Figure 2 Descriptive solution to coin toss

Alternatively, you may prefer a flowchart description to the problem:



GLOSSARY

H . . . 'heads' counter
T . . . 'tails' counter
I . . . '100-toss' loopcounter
Y . . . contains each random number as generated.

Figure 3 Flowchart for coin toss

And, finally, the program:

```
 10 REM**Toss a coin 100 times**
 15 CLS
 20 PRINT "Results from tossing a coin a hundred times:"
 30 PRINT
 40 H = 0
 50 REM **Heads count**
 60 T = 0
 70 REM **Tails count**
 80 I = 1
 90 REM **Number of tosses**
100 Y = INT (2 * RND(1) + 1)
110 IF Y = 2 THEN 160
120 REM **2 means a head**
130 T = T + 1
140 REM **not a head (= a tail)**
150 GOTO 180
160 REM **add 1 to head count**
170 H = H + 1
180 REM **add 1 to no. of tosses**
190 I = I + 1
200 REM **repeat process if less than 100 tosses so far**
```

```
210 IF I <= 100 THEN 90
220 PRINT "Heads   Tails"
230 PRINT H;"    ";T
240 END
```

Program 9 Coin Toss

```
RUN
Results from tossing a coin 100 times:
Heads                   Tails
50                      50
Ready

RUN
Results from tossing a coin 100 times:
Heads                   Tails
42                      58
Ready

RUN
Results from tossing a coin 100 times:
Heads                   Tails
43                      57
```

K Program 9.

Example 2
Write a program to simulate tossing two coins a hundred times. Count and output the number of times that the outcome of this imaginary experiment is:
head-head (HH), tail-tail (TT), and head-tail or tail-head (HT or TH).

Solution
We use the same scoring rules: 1 for a tail and 2 for a head, but we are now tossing two coins. We store the score from the first coin in C1 and the score from the second coin in C2. Then we add C1 and C2 to give the total score for that throw:

S =C1 + C2

S can be 2, 3, or 4:

outcome	score
TT	$1 + 1 = 2$
TH or HT	$1 + 2 = 2 + 1 = 3$
HH	$2 + 2 = 4$

Then we count how many 2's we get, how many 3's and how many 4's.

Counter for 2's	T2
Counter for 3's	M1 (M for mix of H's and T's)
Counter for 4's	H2

The flowchart of the solution is:

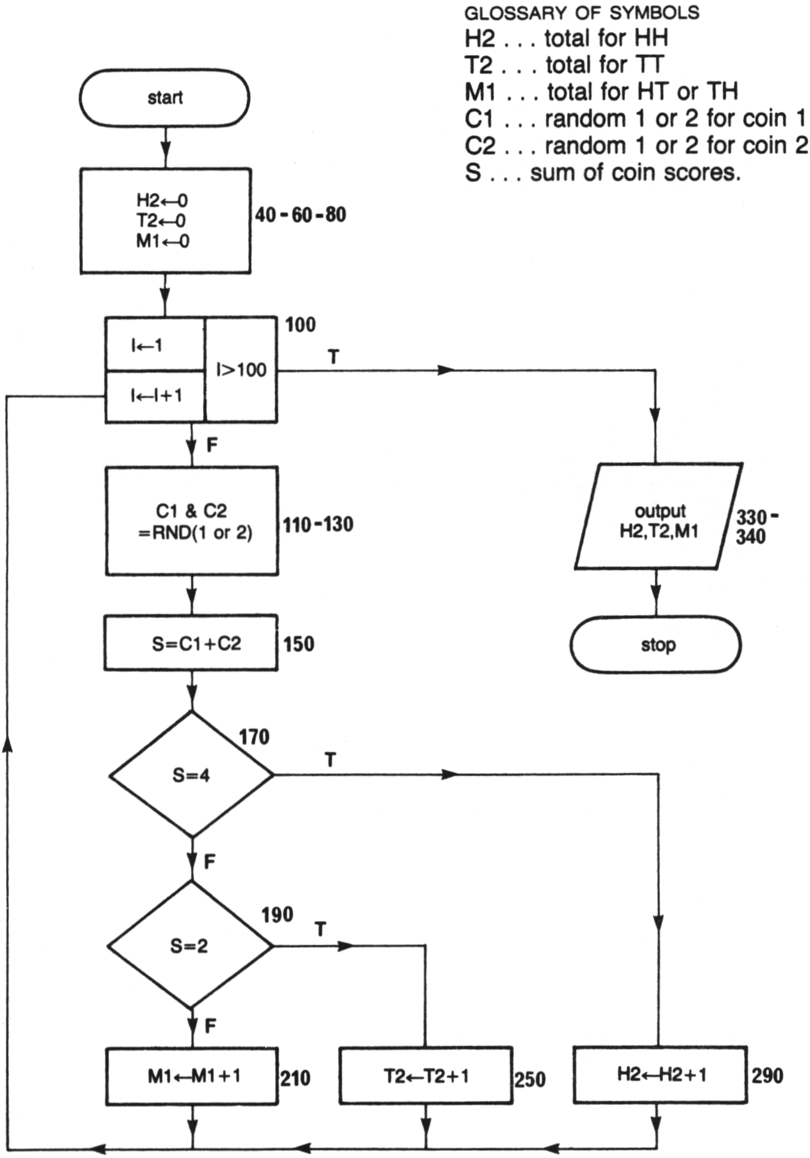


Figure 4 Flowchart for two coin toss

And the program is:

```
 10 REM **Toss 2 coins 100 times**
 15 CLS
 20 PRINT "Results from tossing two coins 100 times:"
 30 PRINT
 40 H2 = 0
 50 REM **"both heads" count**
 60 T2 = 0
 70 REM **"both tails" count**
 80 M1 = 0
 90 REM **"1 head, 1 tail" count**
 95 REM **Start of processing loop**
100 FOR I = 1 TO 100
110 C1 = INT(2 * RND(1) + 1)
120 REM **toss first coin**
130 C2 = INT(2 * RND(1) + 1)
140 REM **toss second coin**
150 S = C1 + C2
160 REM **sum of coin scores**
170 IF S = 4 THEN 280
180 REM **two tails**
190 IF S = 2 THEN 240
200 REM **two heads**
210 M1 = M1 + 1
220 REM **one head, one tail**
230 GOTO 300
240 REM **increase "2 head" count**
250 T2 = T2 + 1
260 REM **exit the loop**
270 GOTO 300
280 REM **increase "2 tail" count**
290 H2 = H2 + 1
300 REM **end of loop**
310 NEXT I
320 REM **output results**
330 PRINT "TT          HT          HH"
340 PRINT T2;"          ";M1;"          ";H2
350 END
```

Program 10 Two coin toss

Here are some typical runs:

```
RUN
Results from tossing 2 coins 100 times:
TT        HT        HH
13        53        34
Ready
RUN

Results from tossing 2 coins 100 times:
TT        HT        HH
33        49        18
Ready
RUN

Results from tossing 2 coins 100 times:
TT        HT        HH
24        55        21
Ready
RUN

Result from tossing 2 coins 100 times:
TT        HT        HH
21        42        37
Ready
```

🄺 Program 10.

6.5 Keeping scores

You may have noticed that we have used some ungainly variable names, such as T2, H2, and M1. Perhaps you have been thinking, 'What about lists? Couldn't they be as useful here as they were with frequency tables?' Indeed they could, so let's try a score list, S(1), for the coin tossing. So we say:

> If the score is 2, add 1 to the number in S(2)
> If the score is 3, add 1 to the number in S(3)
> If the score is 4, add 1 to the number in S(4)

generally

> If the score is S add 1 to the number in S(S)

Application to tossing two coins
If we go back to Example 2, we can re-use the general format, adding our new scoring system:

```
160 S(S) = S(S) + 1
```

The program then becomes:

```
 10 REM **Toss 2 coins 100 times**
 15 CLS
 20 PRINT "Results from tossing two coins 100 times:"
 30 PRINT
 35 DIM S(4)
 40 LET S(4) = 0
 50 REM **"both heads" count**
 60 S(2) = 0
 70 REM **"both tails" count**
 80 S(3) = 0
 90 REM **"1 head, 1 tail" count**
 95 REM **start of processing loop**
100 FOR I = 1 TO 100
110 C1 = INT(2 * RND(1)) + 1
120 REM **toss first coin**
130 C2 = INT(2 * RND(1)) + 1
140 REM **toss second coin**
150 S = C1 + C2
160 S(S) = S(S) + 1
300 REM **end of loop**
310 NEXT I
320 REM **output results**
330 PRINT "TT            HT            HH"
340 PRINT S(2);"            ";S(3);"            ";S(4)
350 END
```

Program 11

```
RUN  (some typical results):
Results from tossing 2 coins 100 times:

TT        HT        HH
30        49        21


Results from tossing 2 coins 100 times:

TT        HT        HH
17        55        28
```

```
Results from tossing 2 coins 100 times:

TT        HT        HH
21        57        22
```

[K] Program 11.

Score lists for dice
The score-list for throwing one die would be:

S(1),S(2),S(3), ..., S(6);

and for throwing two dice:

S(2),S(3),S(4), ...,S(12).

Exercise 1
Write a program to simulate throwing a die a hundred times. Count and output the number of times each score occurs.

Exercise 2
Modify the program written for Exercise 1 to simulate the throwing of two dice a hundred times.

Exercise 3
Write a program to display the data obtained from the program in Exercise 2, in the form of a frequency diagram.

6.6 Concatenation

Having gone to a lot of trouble in Unit 5 to cut up strings, we will now spend some time putting them back together. The second ugliest word in the computing repertoire means to chain or link together. Program 12 shows us what is happening.

```
10 REM ** concatenation **
15 CLS
20 INPUT "First string";A$
30 INPUT "Second string";B$
40 PRINT
50 PRINT A$ + B$
60 END
```

Program 12 Concatenation

```
RUN
First string? micro
Second string? computer
microcomputer

Ready

RUN
First string? concate
Second string? nation
concatenation

Ready
```

SAQ 5

Write a program to input a word and output its plural assuming that all words only need s adding to make their plurals.

Program 13 shows how we can use concatenation to build up a string from a list of symbols. We have stored the letters in A$; in the loop 70-100 we add a new letter to the string on each trip around the loop.

```
 10 REM ** more concatenation **
 15 CLS
 20 REM ** set up directory **
 25 DIM A$(10)
 30 FOR I = 1 TO 10
 35 READ A$(I)
 40 NEXT I
 45 DATA a,b,c,d,e,f,g,h,i,j
 50 REM **********************
 50 C$ = " "
 60 REM ** empty C$ **
 70 FOR J = 1 TO 10
 80 C$ = C$ + A$(J)
 90 PRINT J,C$
100 NEXT J
110 END
```

Program 13 (with ,(tab))

```
 10 REM ** more concatenation **
 15 CLS
 20 REM ** set up directory **
 25 DIM A$(10)
 30 FOR I = 1 TO 10
 35 READ A$(I)
 40 NEXT I
 45 DATA a,b,c,d,e,f,g,h,i,j,
 50 REM *************************
 55 C$ = " "
 60 REM ** empty C$ **
 70 FOR J = I TO 10
 80 C$ = C$ + A$(J)
 90 PRINT J;
 92 IF J < 10 THEN PRINT " ";
 94 PRINT "    ";C$
100 NEXT J
110 END
```

Program 13 (without .)

```
RUN
1        a
2        ab
3        abc
4        abcd
5        abcde
6        abcdef
7        abcdefg
8        abcdefgh
9        abcdefghi
10       abcdefghij
```

K Program 13.

This process is of great value in textual analysis, but we will use it for codes and games.

6.7 STR$

This function has the reverse effect to the `VAL` function. The `VAL` function gives the numerical value of a string, and the `STR$` function turns a number into just a string of characters.

`STR$(X)` gives the string representation of the value of X.

Printing STR$

`STR$ (N)` looks very much like N itself, as the following program shows:

```
10 REM ** the STR$ function **
15 CLS
20 INPUT " Next number?";N
30 PRINT
40 PRINT "01234567890123456789O"
50 PRINT N,STR$(N)
60 END
```

Program 14

```
RUN
Next number? 17
01234567890123456789O
17                17

Ready
RUN
Next number? -17
01234567890123456789O
-17               -17

Ready
RUN
Next number? 99.34
01234567890123456789O
99.34             99.34

Ready
RUN
Next number? -99.34
01234567890123456789O
-99.34            -99.34

Ready
```

[K] Program 14.

However, in each run the second figure is treated as a string.

In the next program (Program 15) we make use of the fact that `STR$ (8)`, say, treats 8 as a string so that we add the character 8 (as opposed to its value) onto the end of a string.

```
10 REM ** more STR$ **
15 CLS
20 LET C$ = " "
30 REM ** empty C$ **
40 FOR J = 1 TO 10
50 C$ = C$ + STR$ J
60 PRINT J,C$
70 NEXT J
80 END
```

Program 15 (with ,)

```
10 REM ** more STR$ **
15 CLS
20 LET C$ = " "
30 REM ** empty C$ **
40 FOR J = 1 TO 10
50 C$ = C$ + STR$(J)
60 PRINT J;
65 IF J < 10 THEN PRINT " ";
68 PRINT "   ";C$
70 NEXT J
80 END
```

Program 15 (without ,)

Output on some microcomputers:

```
RUN
1        1
2        1 2
3        1 2 3
4        1 2 3 4
5        1 2 3 4 5
6        1 2 3 4 5 6
7        1 2 3 4 5 6 7
8        1 2 3 4 5 6 7 8
9        1 2 3 4 5 6 7 8 9
10       1 2 3 4 5 6 7 8 9 10
```

Output on the Oric-1

```
RUN
1        1
2        12
3        123
4        1234
5        12345
6        123456
7        1234567
8        12345678
9        123456789
10       12345678910

Ready
```

Thus on the Oric we can link the characters in adjacent positions.

🄺 Program 15.

Exercise 4
Write a program to input a word from the keyboard, to code each letter as a number, and to output the code as a sequence of numbers.

Guidance if required: set up a directory-list as in Program 15 but for the whole alphabet. Remember the `DIM` statement. Take each letter of the word and compare it with the items of the directory-list. When found in the directory, add that index, in string form, to the code string.

Exercise 5
Write a program to generate 20 random three-letter words. (It's interesting to see how many times you have to run this program to generate a bona fide word.)

Assignment 6

1. Write a program to deal a hand of cards, the size of which is left to you. Print out the hand in the form `1D,KC,8H,6S...`, where `D` = diamonds, `C` = Clubs, etc. 1 = Ace, `T`,`J`,`Q`,`K` stand for ten, jack, queen and king. Remember that when a card has been dealt it cannot be dealt again.

   Guidance if required: write the program in sections:
   - set up the deck (we have previously called it a directory);
   - deal (using the `RND` generator 1 to 52 is the easiest);
   - output.

   When a card has been dealt, put a marker (e.g. an *) in that position to signal that it cannot be used again.

2. Write a program to simulate a game of snakes and ladders using a 4 × 4 board and one dice.

   Guidance if required: though the board is square it can be represented in memory by a list B(I):

   i.e., B(1),B(2),B(3)...B(16)
   B(3) = + 4 could be a ladder going up 4 places.
   B(9) = −7 could be a snake going down 7 places, etc.

   Don't forget that your last throw must give the right number to complete the board at exactly 16.

   Suggestions (not part of the Assignment for your tutor): play the game a few times and estimate the average number of throws needed to run the board. Change the layout of the board and try some more runs. Design a bigger board, etc.

Objectives of Unit 6

When you have finished this unit, check that you are able to:

Use `RND` to generate random numbers between 0 and 1	□
Use `INT` and `RND` to generate integer random numbers between 0 and a given integer *N*	□
Simulate the tosses of a coin	□
Simulate the tosses of two coins	□
Simulate the throw of a die	□
Simulate the throws of two dice	□
Use score lists	□
Concatenate strings	□
Use `STR$(X)`	□

Answers to SAQs and Exercises

SAQ 1

```
10 FOR I = 1 TO 6
20 LET N = 6 * RND(1)
30 PRINT N
40 NEXT I
50 END
```

Program 16

SAQ 2

(a) 4; (b) 9; (c) −3; (d) −1; (e) 1.

SAQ 3

```
RUN
-3.8            -4
-3.6            -4
-3.4            -4
-3.2            -4
-3              -3
-2.8            -3
-2.6            -3
-2.4            -3
-2.2            -3
-2              -2
```

SAQ 4

```
RUN
1.6             1
1.8             1
2               2
2.2             2
2.4             2
2.6             2
2.8             2
3               3
3.2             3
```

Exercise 1

Descriptive algorithm for one die toss

1. Start.
2. Set the six total score locations to zero.
3. Start the 100-throws loop.
4. Generate a random score from the set (1,2,3,4,5,6).
5. Increment the total linked with this score.
6. If loop counter =100 go to statement 4 otherwise go on to statement 7.
7. Output score and total for each score value.
8. Stop.

```
10 REM ** Throw one die 100 times **
15 CLS
20 PRINT "Results from throwing a die 100 times:"
30 PRINT
40 REM ** Space for score counters **
50 DIM S(6)
60 REM ** Zeroise counters **
70 FOR J = 1 TO 6
80 LET S(J) = 0
90 NEXT J
```

```
100 REM ** generation loop **
110 FOR I = 1 TO 100
120 S = INT(6 * RND(1)) + 1
130 S(S) = S(S) + 1
140 REM ** end of loop **
150 NEXT I
160 REM ** output results **
170 PRINT
180 PRINT "Score","Frequency"
190 REM ** display loop **
200 FOR K = 1 TO 6
210 PRINT K,S(K)
220 NEXT K
230 END
```

Program 17 (with ,)

```
 10 REM ** Throw one die 100 times **
 15 CLS
 20 PRINT "Results from throwing a die 100 times:"
 30 PRINT
 40 REM ** Space for score counters **
 50 DIM S(6)
 60 REM ** Zeroise counters **
 70 FOR J = 1 TO 6
 80 S(J) = 0
 90 NEXT J
100 REM ** generation loop **
110 FOR I = 1 TO 100
120 S = INT(6 * RND(1)) + 1
130 S(S) = S(S) + 1
140 REM ** end of loop **
150 NEXT I
160 REM ** output results **
170 PRINT
180 PRINT "Score Frequency"
190 REM ** display loop **
200 FOR K = 1 TO 6
210 PRINT K;"    ";S(K)
220 NEXT K
230 END
```

Program 17 (without ,)

Here are four typical RUNs:

```
RUN

Results from throwing a die
100 times:

Score          Frequency
1              15
2              21
3              14
4              20
5              19
6              11

Ready
RUN

Results from throwing a die
100 times:

Score          Frequency
1              13
2              18
3              24
```

```
4              21
5              16
6               8

Ready
RUN

Results from throwing a die
100 times:

Score          Frequency
1              15
2              25
3              14
4              17
5              15
6              14

Ready
RUN

Results from throwing a die
100 times:

Score          Frequency
1              17
2              12
3              20
4              19
5              18
6              14

Ready
```

🄺 Program 17.

Exercise 2

```
 10 REM ** Throw two dice 100 times **
 15 CLS
 20 PRINT "Results from throwing two dice"
 22 PRINT "100 times."
 30 PRINT
 40 REM ** Space for score counters **
 50 DIM S(15)
 60 REM ** Zeroise counters **
 70 FOR J = 2 TO 12
 80 S(J) = 0
 90 NEXT J
100 REM ** generation loop **
110 FOR I = 1 TO 100
120 S1 = INT(6 * RND(1)) + 1
125 S2 = INT(6 * RND(1)) + 1
130 S = S1 + S2
135 S(S) = S(S) + 1
140 REM ** end of loop **
150 NEXT I
160 REM ** output results **
170 PRINT
180 PRINT "Score    Frequency"
190 REM ** display loop **
200 FOR K = 2 TO 12
205 IF K < 10 THEN PRINT " ";
210 PRINT K;"      ";S(K)
220 NEXT K
230 END
```

Program 18 (without ,)

```
 10 REM ** Throw two dice 100 times **
 15 CLS
 20 PRINT "Results from throwing 2 dice"
 22 PRINT "100 times."
 30 PRINT
 40 REM ** Space for score counters **
 50 DIM S(15)
 60 REM ** Zeroise counters **
 70 FOR J = 2 TO 12
 80 S(J) = 0
 90 NEXT J
100 REM ** generation loop **
110 for i = 1 TO 100
120 S1 = INT(6 * RND(1)) + 1
125 S2 = INT(6 * RND(1)) + 1
130 S = S1 + S2
135 S(S) = S(S) + 1
140 REM ** end of loop **
150 NEXT I
160 REM ** output results **
170 PRINT
180 PRINT "Score","Frequency"
190 REM ** display loop **
200 FOR K = 2 TO 12
210 PRINT K,S(K)
220 NEXT K
230 END
```

Program 18 (with ,)

```
RUN
Results from throwing two dice a hundred times:
Score          Frequency
2              6
3              6
4              6
5              13
6              12
7              21
8              13
9              10
10             6
11             2
12             5
Ready
```

[K] Program 18.

How about 1000 throws? Well, just change 100 to 1000 in lines 22, 10, 22 and 110, and you will get a run such as:

```
RUN
Results from throwing 2 dice a 1000 times:
Score          Frequency
2              29
3              51
4              76
5              120
6              137
7              161
8              140
9              112
10             91
11             58
12             25
Ready
```

Exercise 3

```
 10 REM ** Throw two dice 100 times **
 15 CLS
 20 PRINT "Results from throwing two dice"
 22 PRINT "100 times."
 30 PRINT
 40 REM ** Space for score counters **
 50 DIM S(15)
 60 REM ** Zeroise counters **
 70 FOR J = 2 TO 12
 80 LET S(J) = 0
 90 NEXT J
100 REM ** generation loop **
110 FOR I = 1 TO 100
120 S1 = INT(6 * RND(1)) + 1
125 S2 = INT(6 * RND(1)) + 1
130 S = S1 + S2
135 S(S) = S(S) + 1
140 REM ** end of loop **
150 NEXT I
160 REM ** output results **
170 PRINT
180 REM ** table rows loop **
190 FOR K = 2 TO 12
200 PRINT K;TAB(5);S(K);TAB(10);
210 IF S(K) = 0 THEN 270
220 REM ** column of stars loop **
230 FOR L = 1 TO S(K)
240 PRINT "*";
250 NEXT L
260 REM ** end of stars loop **
270 PRINT
280 NEXT K
290 REM ** end of table loop **
300 END
```

Program 19 (with TAB)

```
 10 REM ** Throw two dice 100 times **
 15 CLS
 20 PRINT "Results from throwing 2 dice"
 22 PRINT "100 times."
 30 PRINT
 40 REM ** Space for score counters **
 50 DIM S(15)
 60 REM ** Zeroise counters **
 70 FOR J = 2 TO 12
 80 S(J) = 0
 90 NEXT J
100 REM ** generation loop **
110 FOR I = 1 TO 100
120 S1 = INT(6 * RND(1)) + 1
125 S2 = INT(6 * RND(1)) + 1
130 S = S1 + S2
135 S(S) = S(S) + 1
140 REM ** end of loop **
150 NEXT I
160 REM ** output results **
170 PRINT
180 REM ** table rows loop **
190 FOR K = 2 TO 12
200 PRINT K;
202 IF K < 10 THEN PRINT " ";
204 PRINT "   ";S(K);
206 IF S(K) < 10 THEN PRINT " ";
```

```
208 PRINT "    ";
210 IF S(K) = 0 THEN 270
220 REM ** column of stars loop **
230 FOR L = 1 TO S(K)
240 PRINT "*";
250 NEXT L
260 REM ** end of stars loop **
270 PRINT
280 NEXT K
290 REM ** end of table loop **
300 END
```

Program 19 (without TAB)

```
RUN
Results from throwing 2 dice 100 times:

2         3         ***
3         9         *********
4         7         *******
5         15        ***************
6         15        ***************
7         13        *************
8         16        ****************
9         5         *****
10        10        **********
11        6         ******
12        1         *

Ready
```

[K] Program 19.

SAQ 5

```
10 REM ** Plurals **
15 CLS
20 A$ = "s"
30 INPUT "enter word";B$
40 PRINT
50 PRINT B$ + A$
60 END
```

Program 20

```
RUN
Enter word? tree
trees

Ready
RUN
Enter word? house
houses

Ready
```

Exercise 4

```
10 REM ** simple code program **
15 CLS
20 PRINT "Simple code generation"
30 C$=" "
40 DIM A$(26)
45 FOR I = 1 TO 26
50 READ A$(I)
55 NEXT I
57 DATA a,b,c,d,e,f,g,h,i,j,k,l,o
58 DATA p,q,r,s,t,u,v,w,x,y,z
```

Set up the directory

```
 60 REM ** directory set up **
 70 INPUT "Next word for coding";W$
 80 L = LEN(W$)
100 REM ** looking up in directory **
110 FOR J = 1 TO L
120 I = 1
125 REM ** matching loop **
130 IF MID$(W$,J,1) = A$(I) THEN 160
140 REM ** exit when letter matched **
150 LET I = I + 1
155 GOTO 125
160 REM ** add index to code in string form **
170 C$ = C$ + STR$(I) + " "
180 NEXT J
190 PRINT
200 PRINT C$
210 END
```

Compare each letter of the word (W$) with each letter in the directory until found

then add the index in string form to the code-string

Program 21

```
RUN
Simple code generation
Next word for coding? computing

3   15   13   16   21   20   9   14   7

Ready
RUN
Simple code generation
Next word for coding? Parliament

16   1   18   12   9   1   13   5   14   20

Ready
RUN
Simple code generation
Next word for coding? Professionals

16   18   15   6   5   19   19   9   15   14   1   12   19
```

🄺 Program 21.

Exercise 5

```
 10 REM ** random 3-letter words **
 15 CLS
 20 PRINT "This program generates random"
 25 PRINT "3-letter words;"
 30 DIM A$(26)
 40 REM ** read in alphabet **
 45 FOR I = 1 TO 26
 50 READ A$(I)
 60 NEXT I
 70 DATA a,b,c,d,e,f,g,h,i,j,k,l,m,n,o
 80 DATA p,q,r,s,t,u,v,w,x,y,z
 90 REM ** reading completed **
100 INPUT "Another list";R$
110 IF R$ <> "Yes" THEN 300
115 REM ** 20 words **
120 FOR K = 1 TO 20
130 W$ = " "
140 REM ** word string empty to start **
150 FOR J = 1 TO 3
160 REM ** start of word **
170 X = INT(26 * RND(1)) + 1
180 W$ = W$ + A$(X)
```

```
190 NEXT J
200 REM ** end of a word **
210 PRINT W$
220 REM ** print the word **
230 NEXT K
240 REM ** go back for next word **
250 GOTO 90
300 END
```

Program 22

```
RUN
This program generates random
three-letter words:
Another list? yes
uyn
uvx
pwe
jky
rrl
kwf
iqg
lah
qhg
ejg
gvn
ojk
ykw
fhk
kwz
tqe
uzh
iip
uzq
ypz
Another list? no

Ready
```

How many times must you run the program to get a 'proper' word?

[K] Program 22.

UNIT 7
Colour, graphics, and sound

Colour, graphics and sound

So far, we have only used the Oric to handle numbers and text, in black and white, to get a sound grounding in the BASIC language. However, some of the more exciting uses of a microcomputer such as the Oric involve using a full range of colours in screen displays, using graphics (picture displays), and in adding sound.

Obviously, arcade-type games are more enjoyable in colour, and with suitable sounds generated when 'invaders' or 'starships' are destroyed. But also, graphs and educational programs, for instance, are clearer and easier to use with the judicious use of colour and, when appropriate, sound.

7.1 Paper colour

There are three main areas of the screen display of which two are under the control of the programmer. The outermost area, called the border, is used for messages (e.g. CAPS when the capitals lock is in use on the keyboard), remains black, and cannot be altered by the programmer using BASIC.
The main part of the screen display, inside the border, is called the paper. This is the area upon which you have seen your programs listed, and your results displayed. The instruction to alter this area's colour is

```
PAPER X
```

where X is a number in the range 0 to 7 inclusive.

The following program will show the effect of the `PAPER` functions:

```
 10 REM ***********************
 20 REM **PAPER COLOURS**
 30 REM ***********************
 35 CLS
 40 FOR X = 0 TO 7
 50 PAPER X
 60 WAIT(100)
100 NEXT X
110 END
```

Program 1 Paper colours

[K] Program 1.

The `WAIT` in line 60 slows down the output display sufficiently to allow you to see the paper colours cycled. Without this line, you will find the rate of change of the colours is too rapid to be seen properly.

You will see the paper area of the screen cycle through eight colours: in order, black, red, green, yellow, blue, magenta, cyan and white.

The function `PAPER` uses the numbers 0 to 7 inclusive to refer to the above colours:

0 black
1 red
2 green
3 yellow
4 blue
5 magenta (light purple)
6 cyan (turquoise blue)
7 white

7.2 Ink colour

The colour of letters drawn on the display screen is known as the ink. Until this unit we have used black ink on white paper; these are the default values – that is to say, the colours provided when the computer is first turned on, or when the existing program is wiped from the memory with the `NEW` function.

As with the `PAPER` function, the colours are referred to by numbers.

```
 10 REM **INK CONTRAST**
 20 CLS
 30 FOR X = 0 TO 23
 40 PRINT "ORIC ink colour demonstration prog."
 50 NEXT X
 60 REM **INK CHANGE**
 70 FOR I = 0 TO 7
 75 FOR P = 0 TO 7
 80 INK I
 85 PAPER P
 90 WAIT(100)
 95 NEXT P
100 NEXT I
110 INK 0
115 PAPER 7
120 END
```

Program 2 Ink/paper contrast

[K] Program 2.

You will see from running Program 2 that all possible combinations of paper and ink colour are cycled through. Naturally, when paper and ink colours are the same you will not actually see any words!

Some combinations of ink and paper are easier to read than others. Worse still, unless you are fortunate enough to have a colour monitor to use, instead of an ordinary colour television, with your computer, you will suffer a loss of picture quality with some colour combinations, which makes the letters appear to 'scintillate' – to move. It is important, therefore, to choose colour combinations wisely and sparingly. You will soon discover which combinations of colours suit your displays the best, by trial and error. You could make a start by using Program 2, and noting which combinations of paper and ink stand out clearly.

You will also need to bear in mind that, if you hope to produce programs for use by other people, not everyone will be using a colour display screen. Colour combinations of ink and paper need to have equally good contrast in black and white. For instance, blue and red are clearly distinguishable in colour, but have very poor contrast in black and white, being right next to each other in the contrast scale. If you turn your colour television or monitor to produce a black and white picture, you will be able to see if you have allowed sufficient contrast.

Some colour microcomputers have the disadvantage that the display screen must be cleared (generally using `CLS`) before colour changes can be made. This is particularly a problem when it has taken some time to produce a complex colour display. The Oric does not need to clear the screen; you will have seen from Program 2 that ink and paper colour changes are instantaneous. However it does mean that if you change paper halfway through a program, the whole screen will alter, not merely the part referred to. The same happens if an ink colour change is called up.

The way that this is overcome is to use special control characters which refer to each individual line of the screen; although the screen is 40 characters wide, only

38 lines are usually written upon by program listings and output. The other two character positions at the left-hand end of each line hold what are known as the attributes of that line. The attributes are the colour of the ink and paper on that line, whether the line should be inverted (ink and paper colours swapped over), whether the line should flash or stay still, the size of the characters, etc.

At the end of Unit 3 we produced a list of ASCII characters produced by the Oric computer. You may remember that we said there were problems when running the program provided between characters 126 and 161. This is because the unused ASCII codes in this range have been used to hold the attribute codes so that they can be used in PRINT statements in programs. Some of the attributes allow the changing of television picture synchronisation for use on sets with different characteristics from those exhibited by British televisions. This is what causes the picture to go haywire when running the program in that range.

Anyway, the colour attributes are stored in the following codes:

Black	144
Red	145
Green	146
Yellow	147
Blue	148
Magenta	149
Cyan	150
White	151

for background (paper) colours, and

Black	129
Red	130
Green	131
Yellow	132
Blue	133
Magenta	134
Cyan	135
White	136

for foreground (ink) colours.

These attributes are called up using the PRINT statement:

```
PRINT CHR$(A)
```

where A is the ASCII code; for example PRINT CHR$(145) for a red line. The following program uses the ASCII codes for the colour attributes to draw a six-colour straight 'rainbow' (well, you can't have everything at once!).

```
10 REM **Colours**
15 CLS
17 PRINT "Oric's six-colour rainbow!"
20 FOR I = 1 TO 6
30 READ A
40 PRINT CHR$(A)
50 NEXT I
60 END
70 DATA 145,147,146,148,150,149
```

Program 3

[K] Program 3.

7.3 Coloured light

Now that we have an insight to the way that the Oric deals with colour, let us see how a colour television or monitor produces the colours we want. Run Program 4, 'Coloured light', and you will have a visual demonstration of the explanation which is to follow.

We have used the `PLOT` statement, introduced in Unit 5, to draw three intersecting coloured spotlights onto the screen. When using `PLOT`, the format is

```
PLOT X,Y,A
```

where X is the x-co-ordinate of the point to be plotted (the character position across the screen) and Y is the y-co-ordinate of the point (the line number down the screen). A is the attribute, 16 means black, 17 means red, 18 – green, 19 – yellow, 20 – blue, 21 – magenta, 22 – cyan, and 23 – white. There is the added complication that

```
PLOT X,Y,A
```

means 'all character positions from character X on line Y to the right hand end of that line should be affected by attribute A'. In other words, a coloured stripe, starting at character X and going to the end of line Y, will be produced. The way that precise control is obtained is by colouring over the unwanted part of the line with the background colour:

```
PAPER 0
PLOT X,Y,17
PLOT X + 1,Y,16
```

would produce a black background, then a red stripe, then black out all but one character position of the stripe.

This is how we have produced the display for Program 4.

```
 10 REM **SPOTLIGHTS**
 20 CLS
 30 PAPER 0
 40 REM ******************************
 50 FOR Y = 2 TO 4
 60 PLOT 1,Y,17
 70 PLOT 21,Y,16
 80 NEXT Y
 90 REM ******************************
100 FOR Y = 5 TO 11
110 PLOT 1,Y,17
120 PLOT 13,Y,19
130 PLOT 21,Y,18
140 PLOT 33,Y,16
150 NEXT Y
160 REM ******************************
170 FOR Y = 12 TO 18
180 PLOT 1,Y,17
190 PLOT 5,Y,21
200 PLOT 13,Y,23
210 PLOT 21,Y,22
220 PLOT 25,Y,18
230 PLOT 33,Y,16
235 NEXT Y
240 REM ******************************
250 FOR Y = 19 TO 23
260 PLOT 1,Y,16
270 PLOT 5,Y,20
```

```
280 PLOT 13,Y,22
290 PLOT 25,Y,18
300 PLOT 33,Y,16
310 NEXT Y
320 REM ******************************
330 FOR Y = 24 TO 26
340 PLOT 1,Y,16
350 PLOT 5,Y,20
360 PLOT 25,Y,16
370 NEXT Y
380 REM ******************************
390 END
```

Program 4 Coloured light

🄺 Program 4.

You will see that the program produces three overlapping rectangular spotlights – the left is red, the right one is green, and the bottom one is blue. Where the red and green lights meet they produce yellow; the blue and green produce cyan (light blue) and the red and blue make magenta (purple), where there is not light we get black.

A colour tube is sensitive to red, green and blue light output, and produces other shades by overlapping (or mixing, if you like) tiny dots of these three primary colours.

If you look at the paper colour numbers, you will see how useful they can be. Blue is 4, red is 1; blue and red make magenta, which is 5 – so 4 + 1 = 5. Likewise, red (1) and green (2) make yellow (3). Green (2) and blue (4) make cyan (7). Blue (4), red (1) and green (2) make white (7). Black and any colour just gives the colour itself, so black must therefore be 0. Magenta (5) and green (2) make white (7); cyan (6) and red(1) make white (7); yellow (3) and blue (4) make white (7). Here ends the colour theory(!). This can be useful if you wish to use calculations to decide upon colour displays.

SAQ 1
Which colours will be generated by these instructions:

(a) `PAPER 3`
(b) `INK 4`
(c) `PLOT 5,3,21`

Example 1
Now that we have dealt with low resolution or character graphics in colour, we can produce a kaleidoscope. A kaleidoscope is a device which produces symmetrical patterns which change colour and shape in a random manner.

Now we need to develop a strategy to solve the programming problem. If we want to plot a point on the screen we use

```
PLOT X,Y,C
```

where X and Y are the co-ordinates of the point and C is its colour.

Let us make our kaleidoscope have symmetry about a line which splits the screen in two vertically, and about a line which splits the screen in two horizontally. This means that any point X,Y we plot on the screen will have a reflection at 39 – X,Y (not 40, to avoid writing over the attribute column at the far left of the screen). These two points will have reflections at X,25 – Y and at 39 – X,25 – Y so

```
PLOT X,Y,C
PLOT 39 - X,Y,C
PLOT X,25 - Y,C
PLOT 39 - X,25 - Y,C
```

will form the 'nucleus' of our program.

We now need to make X,Y and C change at random.

`X = RND(1) * 38` will keep the coloured square from going off the screen in the horizontal (X) direction.

`X = RND(1) * 25` will keep the coloured square from going off the screen in the vertical (Y) direction.

`C = 17 + RND(1) * 3` will make the square change colour randomly choosing among red, green, yellow and blue.

All we need to do is to start off with a blank, black screen (`CLS` and `PAPER 0`) and branch back to select a new kaleidoscope position, and we are in business.

```
 3 REM **KALEIDOSCOPE**
 5 CLS
 7 PAPER 0
10 X = RND(1) * 38
20 Y = RND(1) * 25
30 C = 17 + RND(1) * 3
40 PLOT X,Y,C
42 PLOT 39 - X,Y,C
44 PLOT X,25 - Y,C
46 PLOT 39 - X,25 - Y,C
50 GOTO 10
```

Program 5 Kaleidoscope

The pattern becomes more symmetrical as time goes on, because we are unable to limit the colours to single character positions, as this would destroy the pattern to the right of new characters plotted. Touch `CTRL` and `C` to break out of this program.

[K] Program 5.

Example 2

Another type of pattern that can effectively use colour graphics is a simple flag design. We can produce a vertical tricolore (as per the reversed French flag).

We come back to the `PLOT` statement

```
PLOT X,Y,A
```

as the centre of our program.

We need to repeat the flag pattern on each line of the screen display, so our `PLOT` statement should be sandwiched inside a `FOR ... NEXT ...` loop:

```
60 FOR Y = 2 TO 25
70 PLOT X,Y,A
80 NEXT Y
```

We need to split up the screen into three sections, vertically, to make the flag:

```
40 FOR X = 0 TO 38 STEP STEP 13
90 NEXT X
```

and we need to change the colour in each section:

```
 50 READ A
110 DATA 17,23,20
```

(in this case red, white and blue).

```
 10 REM **Vertical Tricolore**
 20 CLS
 30 PAPER 0
 40 FOR X = 0 TO 38 STEP 13
 50 READ A
 60 FOR Y = 2 TO 25
 70 PLOT X,Y,A
 80 NEXT Y
 90 NEXT X
100 END
110 DATA 17,23,20 - colour numbers
```

Program 6 Vertical Tricolore flag

[K] Program 6.

Don't forget the colour numbers:

16 = black, 17 = red, 18 = green, 19 = yellow, 20 = blue, 21 = magenta, 22 = cyan, 23 = white.

SAQ 2

(a) Convert Program 6 to draw the French flag (blue/white/red vertical stripes).
(b) Convert Program 6 to draw the Belgian flag (black/yellow/red vertical stripes). Note: use `PAPER 7` here to give a contrasting background.
(c) Convert Program 6 to draw the Italian flag (green/white/red vertical stripes). Note: change background back to `PAPER 0`.

7.4 High resolution colour

We have so far limited ourselves to low resolution or character graphics. The Oric is also capable of high resolution or dot graphics. The screen area is broken up into tiny dots, called pixels, which is short for picture elements. In high resolution graphics, each pixel is addressable, so giving us very fine detail when required.

We call up the extra facilities by adding a new line at the start of our program:

```
10 HIRES
```

This changes the way the computer displays on the screen. A small 'window' three lines high is provided at the base of the screen to view program instructions. However, the listing is cycled at such high speed that it is best to break out of `HIRES` by typing `TEXT` then touching `RETURN` as a command, which will clear the screen back to full text handling. `LIST` will then produce a normal listing.

We shall now look at some of the high resolution instructions available with the ORIC. The screen is defined as 240 pixels wide (numbered 0 to 239) and 200 pixels high (numbered 0 to 199).

The instruction `CURSET` sets the cursor to an absolute position.

```
CURSET X,Y,FB
```

where X is the X position (0 to 239) and Y is the Y position (0 to 199) and FB is the foreground/background code. FB should be 0 for background colour, 1 for foreground colour, 2 to invert colours, and 3 to do nothing (null).

The instruction `CIRCLE` draws a circle centred at the current cursor position:

```
CIRCLE R,FB
```

where R is the radius of the circle and FB is the foreground/background code, as above. Care must be taken that no part of the circle leaves the screen.

Example 3
We can use the circle drawing facility to draw a Japanese flag (red circle on a white background).

We need to set the cursor to the centre of the screen: the X value is 240/2 = 120: the Y value is 200/2 = 100. FB is 3, as we do not wish to alter the colour we choose.

If we put the `CIRCLE` instruction inside a `FOR ... NEXT ...` loop, we can effectively 'fill in' the circle by altering the radius from 1 to the required maximum value (in this case 50 looks about right).

Here is the program in full:

```
10 REM **JAPANESE FLAG**
20 HIRES
30 INK 1
40 PAPER 7
50 CURSET 120,100,3
60 FOR X = 1 TO 50
70 CIRCLE X,1
80 NEXT X
90 END
```

Program 7 Japanese Flag

[K] Program 7.

Just as a nice colour demonstration, try out this program. The `GOSUB` statement is covered in Unit 9; suffice it to say it saves repeated keying in of the `CIRCLE 40,1` instruction in line 200.

```
  5 HIRES
  7 PAPER 7
 10 CURSET 120,60,3
 20 GOSUB 200
 30 CURSET 60,60,3
 40 GOSUB 200
 50 CURSET 180,60,3
 60 GOSUB 200
 70 CURSET 90,120,
 80 GOSUB 200
 90 CURSET 150,120,
100 GOSUB 200
120 FOR X = 0 TO 6
130 INK X
140 WAIT(50)
150 NEXT X
155 GOTO 120
160 END
200 CIRCLE 40,1
210 RETURN
```

Program 8 Olympic symbols

[K] Program 8.

FILL
Another useful high resolution graphics instruction is `FILL`. It has the following format:

```
FILL R,C,N
```

and fills C character cells by R rows with the value in N; there are 200 rows (1 pixel deep) and 40 characters (6 pixels wide). It is useful to fill up rectangular areas of the screen in particular.

N can be a colour number (e.g. 17 = red, as before). If we divide up the screen horizontally in three sections, we can produce horizontal tricolore flags. In the following program, X is the colour number, N is the row number (0 to 199) and A is the counter to divide the screen into three.

We have included the line

```
60 X = B + 16
```

so that we can read in the paper colour numbers (B) as data, rather than the attribute numbers (16-23). The `FILL` instruction is

```
70 FILL 1,1,X
```

because we are filling one character cell on one row with the colour specified by the number X at one time. The nested `FOR ... NEXT ...` loops ensure that every cell is filled in succession.

```
 10 REM **Fill: Tricolore**
 20 HIRES
 30 FOR A = 0 TO 2
 40 READ B
 50 FOR N = A * 66 TO (A+1) * 66
 60 X = B + 16
 70 FILL 1,1,X
 80 NEXT N
 90 NEXT A
100 END
110 DATA 1,7,4      Red, white, blue (The Netherlands)
```

Program 9 Horizontal tricolore flag

$\boxed{K}$ Program 9.

Exercise 1
We have just produced the flag of the Netherlands, in Program 9. Change the `DATA` statements to produce

(a) the German flag (black/red/yellow stripes)
(b) the flag of Sierra Leone (green/white/blue stripes)
(c) the flag of Austria (red/white/red stripes).

Exercise 2
Try to adapt the 'Tricolore' program – Program 9 – to produce a full screen version of the rainbow shown in Program 3. The colour order is red, yellow, green, blue, cyan, magenta. Line 30 needs changing to produce six stripes instead of three; line 50 needs altering in the same way.

Now for the moment of truth! Let us go the whole way and produce a seven-colour rainbow. As you may have noticed from the previous high resolution programs, we have very fine colour control, down to an element one pixel high by six pixels wide. By altering pixel rows of alternating colours we can make a possible attempt to display colours not provided automatically by the computer.

For example, if we alternate rows of red and yellow, we can produce a passable

orange colour. If we now produce a program with seven colour stripes, each made up of two alternating colours, we have the basis of our 'real' rainbow. For the red, yellow, green, blue, cyan and magenta stripes, we make both of the alternating colours the same, as we want the 'pure' colour. For the orange stripe, we use red and yellow for the two alternating colours.

```
  5 REM **SEVEN-COLOUR RAINBOW**
 10 HIRES
 20 FOR C = 0 TO 6
 25 READ A,B
 30 FOR X = 14 * C TO 14 * (C + 1)
 50 FILL 1,1,A
 60 FILL 1,1,B
 80 NEXT X
 90 NEXT C
100 END
110 DATA 17,17
120 DATA 17,19
130 DATA 19,19
140 DATA 18,18
150 DATA 20,20
160 DATA 22,22
170 DATA 21,21
```

Program 10 Seven-colour rainbow (full screen)

K Program 10.

7.5 Low resolution graphics

It is possible to define your own characters with the Oric, which you could colour and move around with the statements we have already described, as per space games commercially available. For detailed drawing on screen, there is one further instruction we have not mentioned:

`DRAW X,Y,FB`

which draws a line from the present cursor position to that position plus X,Y. It is the same as the `CURMOV` statement in format.

It is not within the scope of this course to go into the details of producing user-defined characters. However, we can illustrate the method of producing moving graphics with the use of low resolution screen display.

We have so far used two of the Oric's graphics modes: `TEXT` for characters and `HIRES` for the most detailed graphics. There are two remaining modes; `LORES 0` and `LORES 1`. `LORES 0` uses the standard alphanumeric character set, and `LORES 1` uses the alternative graphics (picture) and character set. In both cases the background is cleared to black.

You can use the following program to produce, and then select, graphics characters to make your own pictures:

```
10 REM **GRAPHICS character set**
20 FOR X = 32 TO 128
30 PRINT X;"    ";CHR$(27);"I";CHR$(X)
40 PRINT
50 WAIT(100)
60 NEXT X
70 END
```

Program 11 Graphics character set

Ⓚ Program 11.

Example 4: Moving bus

This program will draw what is, with the eye of faith, a bus made out of alphabetic characters, near the left-hand side of the display screen:

```
10 X = 1
19 PLOT X,5," FTTTL"
20 PLOT X,6," HHHHH"
21 PLOT X,7,"  0 0"
```

Program 12 Bus

Ⓚ Program 12.

Now change line 10 to:

```
10 FOR X = 1 TO 32
```

and add line 30:

```
30 NEXT X
```

which will give the following listing:

```
10 FOR X = 1 TO 32
19 PLOT X,5," FTTTL"
20 PLOT X,6," HHHHH"
21 PLOT X,7,"  0 0"
30 NEXT X
35 END
```

Program 13 Moving bus

Ⓚ Program 13.

Now run your Program 13. Magic! The bus moves rapidly across the screen. What happens is that in the `FOR .. NEXT ...` loop, every time 1 is added to the value of X, the bus is redrawn one position to the right of the previous drawing. It is important to leave one line of spaces at the left-hand end of the drawing, as this causes the drawing to 'rub out' the previous drawings as the motion across the screen occurs.

This is exactly the way that cartoon drawings are made to 'move' – the slight difference in position of each drawing, coupled with the rapid movement from one picture to the next, makes us think the picture is actually moving.

We may adjust the speed of the bus by adding a `WAIT` instruction at line 25; the larger the number after `WAIT`, the slower and jerkier will be the bus's motion. Try adjusting the `WAIT` statement until you think the motion looks about right.

If we now add

```
4 FOR X = 1 TO 38
5 PLOT X,8,"="
6 NEXT X
```

then we have added a road for the bus to drive along.

```
 1 CLS
 4 FOR X = 1 TO 38
 5 PLOT X,8,"="
 6 NEXT X
10 FOR X = 1 TO 32
19 PLOT X,5," FTTTL"
20 PLOT X,6," HHHHH"
21 PLOT X,7," O O"
25 WAIT(10)
30 NEXT X
35 END
```

Program 14 Moving bus – adjustable speed plus road

🄺 Program 14.

Using the principles already explained, you can add colour to the picture, as required.

We can now take this one step further by replacing the alphanumeric characters in the drawing with graphic characters. We can obtain graphics characters when text is usually produced by using the ASCII code CHR$(9) to switch to graphics and CHR$(8) to switch to standard character set.

This program draws a graphics lorry and drives it across the screen:

```
100 CLS
110 LORES 0
115 INK 2
120 A$ = CHR$(9) + " £££££," + CHR$(8)
130 B$ = CHR$(9) + " £££££," + CHR$(8)
140 C$="      O    O"
150 FOR X = 1 TO 38
160 PLOT X,8,"="
170 NEXT X
180 FOR X = 1 TO 31
190 PLOT X,5,A$
200 PLOT X,6,B$
210 PLOT X,7,C$
220 WAIT (8)
230 NEXT X
240 END
```

Program 15 Graphics lorry

🄺 Program 15

Now experiment with the graphics symbols to make your own pictures. Don't forget that blank line at the left-hand side of the picture.

7.6 Sound in programs

If you have been used to the faintly audible sound effects produced by some microcomputers then the Oric is certainly a big improvement. It has a special sound chip with three tone and one noise channels, and variable volume, producing a high sound output, if required.

When you first turn the power on the Oric makes a high-pitched noise each time a key is touched, and a lower-pitched noise when a control key or RETURN is touched. This can be switched off using CTRL and F; a second touching of CTRL-F will bring the sounds back.

If you like video games then the Oric has four predefined sounds which will help you achieve realistic effects:

1. `ZAP` which sounds like a space gun;
2. `PING` which sounds like a bell (also available on CTRL-G);
3. `SHOOT` which sounds like gunfire; and
4. `EXPLODE` which sounds like an explosion.

Each of these effects is obtained by typing in the word, in capitals, and then touching RETURN. You can, of course, include them as statements in your programs.

MUSIC

The `MUSIC` function has been designed to provide pure sounds for keying in standard music. We shall concentrate on showing how simple melody can be produced although, of course, with three channels, complex harmonies can be programmed when you are totally familiar with the sound instructions.

The `MUSIC` statement has this form:

```
MUSIC C,T,P,V
```

where `C` is the channel (1, 2 or 3 tone channel)
`T` is the octave of the note (0 to 6, with 0 as the lowest)
`P` is the pitch of the note (semitones 1 to 12: please see the accompanying table)
and V is the volume (1 to 15 fixed volume levels)

Pitch of notes, within each octave

Semitone no.	Note name
1	C
2	C sharp
3	D
4	D sharp or E flat
5	E
6	F
7	F sharp
8	G
9	G sharp or A flat
10	A
11	A sharp or B flat
12	B

`PLAY` is the instruction which controls the actual production of the notes. It is needed twice; once to switch the music on and a second time to switch it off. You will find, to your cost, that if you break into a music program with CTRL-C when sounds are actually being produced, the sounds will continue on the last note output until you key in `PLAY 0,0,0,0` then touch RETURN.

`PLAY` allows you to do some clever things, musically speaking, but we will limit ourselves in our examples to

```
PLAY C,0,0,0,0
```

where C tells us which channel is being used (1, 2 or 3, as in the `MUSIC` instruction).

We will use

PLAY 0,0,0,0

to turn off the music when we have finished.

Now let us familiarise ourselves with the MUSIC instruction.This program plays one hundred notes, all pitched at G in the third octave; each note is shorter than the last, varying from a value of 100 to 1, as specified by the WAIT statement in line 160 to illustrate the control over duration of a note:

```
 10 REM ** LENGTH OF NOTES **
 20 REM *******************************
 30 REM N = NUMBER OF NOTES
 40 REM V = VOLUME
 50 REM P = PITCH OF NOTE
 60 REM D = DURATION OF NOTE
 70 REM T = OCTAVE OF NOTE
 80 REM C = CHANNEL
 90 REM *******************************
100 LET V = 6
105 LET T = 3
110 LET C = 1
115 LET P = 8
120 REM *******************************
130 FOR N = 1 TO 100
140 MUSIC C,T,P,V
150 PLAY C,0,0,0
155 LET D = 100 / N
160 WAIT D
170 PLAY 0,0,0,0
180 NEXT N
190 REM *******************************
```

Program 16 Length of notes

[K] Program 16

The PLAY statement in line 150 merely puts into action the sound specified by the MUSIC statements in line 140. That sound would carry on indefinitely were it not for the WAIT statement in line 160, which is a timed delay. Once the delay has expired, control passes to line 170 which stops the sound.

This program, by contrast, keeps all its notes of the same duration, but plays all pitches from C in octave 0 to B in octave 6, a total of seven octaves each of 12 semitones; 84 notes in all.

```
 10 REM ** PITCH OF NOTES **
 20 REM *******************************
 30 REM ** 7 OCTAVES OF 12 SEMITONES **
 40 REM V = VOLUME
 50 REM P = PITCH OF NOTE
 60 REM D = DURATION OF NOTE
 70 REM T = OCTAVE OF NOTE
 80 REM C = CHANNEL
 90 REM *******************************
100 LET V = 6
110 LET C = 1
115 LET D = 100
120 REM *******************************
125 FOR T = 0 TO 6
130 FOR P = 1 TO 12
140 MUSIC C,T,P,V
150 PLAY C,0,0,0
```

```
160 WAIT D
170 PLAY 0,0,0,0
180 NEXT P
185 NEXT T
190 REM ******************************
```

Program 17 Pitch of notes

[K] Program 17

In western music, there are eight notes in a scale, but there are twelve semitones in each octave (as shown in Figure 1). The computer's octaves refer to the key of C major, so for other scales, some notes will be in one octave and some in the next, so that DATA statements need to contain the pitch number (P), the duration of the note (D) and the octave of the note (T); we have stuck to this notation in our examples. The next program plays the scale of G (in the third/fourth octaves).

```
 10 REM ** SCALE OF G **
 20 REM ******************************
 30 REM N = NUMBER OF NOTES
 40 REM V = VOLUME
 50 REM P = PITCH OF NOTE
 60 REM D = DURATION OF NOTE
 70 REM T = OCTAVE OF NOTE
 80 REM C = CHANNEL
 90 REM ******************************
100 LET V = 6
110 LET C = 1
120 REM ** START OF SCALE **
130 FOR N = 1 TO 8
140 READ P,D,T
150 MUSIC C,T,P,V
160 PLAY C,0,0,0
170 WAIT D
180 PLAY 0,0,0,0
190 NEXT N
200 REM ** END OF SCALE: DATA NEXT **
210 DATA 8,100,3
220 DATA 10,100,3
230 DATA 12, 100,3
240 DATA 1,100,4
250 DATA 3,100,4
260 DATA 5,100,4
270 DATA 7,100,4
280 DATA 8,100,4
290 REM ** END OF DATA **
```

Program 18 Scale of G

[K] Program 18.

The scale of G goes G,A,B,C,D,E,F sharp, G flat (hence the semitone numbers in the DATA statements, from Figure 1).

Now, let us play a tune. Each set of three numbers in the DATA statements refers to one note. We have made 100 as the value of a minim, so that 50 is a crotchet and 25 a quaver. Care must be taken in keying in the DATA as with any program statements.

```
10 REM ** LONDON'S BURNING **
20 REM ** IN THE KEY OF G MAJOR **
25 REM ******************************
30 REM N = NUMBER OF NOTES
40 REM V = VOLUME
50 REM P = PITCH OF NOTE
60 REM D = DURATION OF NOTE
70 REM T = OCTAVE OF NOTE
```

```
 80 REM C = CHANNEL
 90 REM ******************************
100 LET V = 6
110 LET C = 1
120 REM ** START OF TUNE **
130 FOR N = 1 TO 28
140 READ P,D,T
150 MUSIC C,T,P,V
160 PLAY C,0,0,0,0
170 WAIT D
180 PLAY 0,0,0,0
190 NEXT N
200 REM ** END OF TUNE: DATA NEXT **
210 DATA 3,25,3,3,25,3,8,50,3,8,50,3
220 DATA 3,25,3,3,25,3,8,50,3,8,50,3
230 DATA 10,25,3,10,25,3,12,50,3,12,50,3
240 DATA 10,25,3,10,25,3,12,50,3,12,50,3
250 DATA 3,50,4,3,100,4,3,50,4,3,100,4
260 DATA 3,25,4,1,25,4,12,50,3,12,50,3
270 DATA 3,25,4,1,25,4,12,50,3,12,50,3
```

Program 19 London's burning

[K] Program 19.

Why not play around with this program to get used to the variables? Change the volume (set in line 100) – a larger number will make a louder sound and a smaller number a quieter one. If you change the channel number (line 110) to either 2 or 3, it makes no difference as channels 1, 2 and 3 are identical. Do not alter line 130 as 28 is the number of notes in the tune – changing this will upset the READ statement in line 140. You can alter the tempo of the music by altering line 170. If you made it

```
170 WAIT (2 * D)
```

the music would be twice as slow, and if it were to be

```
170 WAIT (D / 2)
```

it would be twice as fast, and so on.

```
 10 REM ** DARBY KELLY **
 20 REM ** IN THE KEY OF G MAJOR **
 25 REM ****************************
 30 REM N = NUMBER OF NOTES
 40 REM V = VOLUME
 50 REM P = PITCH OF NOTE
 60 REM D = DURATION OF NOTE
 70 REM T = OCTAVE OF NOTE
 80 REM C = CHANNEL
 90 REM ****************************
100 LET V = 6
110 LET C = 1
120 REM ** START OF TUNE **
130 FOR N = 1 TO 102
140 READ P,D,T
150 MUSIC C,T,P,V
160 PLAY C,0,0,0,0
170 WAIT D
180 PLAY 0,0,0,0
190 NEXT N
200 REM ** END OF TUNE: DATA NEXT **
210 DATA 3,25,3,8,50,3,12,25,3,12,50,3
220 DATA 3,25,3,8,50,3,12,25,3,12,25,3
```

```
230 DATA 3,25,3,8,50,3,12,25,3,12,17,3,10,17,3,12,17,3
240 DATA 1,50,4,10,25,3,10,50,3,3,25,3
250 DATA 8,50,3,12,25,3,12,17,3,10,17,3,8,17,3
260 DATA 5,50,3,1,25,4,1,50,4,5,25,3
270 DATA 3,50,3,12,25,3,12,50,3,10,25,3
280 DATA 10,50,3,8,25,3,8,50,3
290 DATA 12,25,3,10,50,3,3,25,3,3,50,3
300 DATA 12,25,3,10,50,3,3,25,3,3,50,3
310 DATA 12,25,3,10,50,3,3,25,4,3,17,4
320 DATA 2,17,4,3,17,4,3,50,4,3,25,3,3,200,3
330 DATA 1,25,4,12,50,3,10,25,3,8,50,3
340 DATA 7,25,3,5,50,3,8,25,3,8,17,3
350 DATA 7,17,3,5,17,3,3,50,3
360 DATA 12,25,3,12,50,3,10,25,3,10,50,3
370 DATA 8,25,3,8,50,3,12,25,3,10,50,3
380 DATA 3,25,3,3,50,3,12,25,3,10,50,3
390 DATA 3,25,3,3,50,3,12,25,3,10,50,3
400 DATA 3,25,4,3,17,4,2,17,4,3,17,4,3,50,4
410 DATA 3,25,3,3,200,3,1,25,4,12,50,3
420 DATA 10,25,3,8,50,3,7,25,3,5,50,3
430 DATA 8,25,3,8,17,3,7,17,3,5,17,3
440 DATA 3,50,3,12,25,3,12,50,3,10,25,3
450 DATA 10,50,3,8,25,3,8,25,3
```

Program 20 Darby Kelly

Try adjusting tempo via line 170 e.g. `170 WAIT 4 * D / 7`

[K] Program 20.

If you run this program, you will see that quite complex tunes can be played even without venturing into variations of volume or using harmony. Triples are no problem: refer to line 230. If three notes, forming a triple, are equal to a crotchet (50) then each is worth 17 (approximately). If you are well versed in musical theory you should have no difficulty in writing music programs. If not, a book of simple tunes, such as those written for children learning to play the recorder, would give suitable tunes to start with. Write the name of each note (A,B,C etc.) under each note in the piece of music, then refer to Figure 1 to decide the semitone number.

Exercise 3
Try to adapt Program 18 to play the scale of C (third octave), instead of G. (Note: you need only change the DATA statements).

Assignment 7

1. Using the techniques shown in 7.5, Low resolution graphics, design a vehicle made up from graphics characters, and write a program to make it move. Enhance the display by the addition of a roadway etc. Add colour where suitable and also sound effects if appropriate.
2. Choose a simple tune and write a program to get the computer to play it. If possible send your tutor a copy of the music as well as the program.

Objectives of Unit 7

When you have finished this unit check that you are able to:

- Use PAPER and INK to control the colour of outputs to the screen ☐
- Describe how coloured lights combine to make other colours ☐
- Add colour to programs to improve their outputs ☐
- Produce simple moving graphics pictures ☐
- Use PLOT, DRAW, CURMOV, CURSET, CIRCLE and FILL to make pictures ☐
- Use MUSIC and PLAY to play tunes ☐
- Use ZAP, SHOOT, PING and EXPLODE to add sound effects ☐

Answers to SAQs and Exercises

SAQ 1

(a) yellow screen
(b) blue writing on the screen
(c) plot a magenta stripe, starting at character position 5 on line 3

SAQ 2

(a) Change line 110 to: `110 DATA 20,23,17`

(b) Change line 110 to: `110 DATA 16,19,17`
and line 30 to: `30 PAPER 7`

(c) Change line 30 back to: `30 PAPER 0`
and line 110 to: `110 DATA 18,23,17`

Exercise 1

(a) Change line 110 to: `110 DATA 0,1,3`

(b) Change line 110 to: `110 DATA 2,7,4`

(c) Change line 110 to: `110 DATA 1,7,1`

Exercise 2: Full screen rainbow

```
 10 REM ** Oric's 6-colour rainbow **
 20 HIRES
 30 FOR A = 0 TO 5
 40 READ COLOUR
 50 FOR N = A * 33 TO (A + 1) * 33
 60 FILL 1,1COLOUR
 70 NEXT N
 80 NEXT A
 90 END
100 DATA 17,19,18,20,21,22
```

Exercise 3: The Scale of C

```
 10 REM ** SCALE OF C **
 20 REM ******************************
 30 REM N = NUMBER OF NOTES
 40 REM V = VOLUME
 50 REM P = PITCH OF NOTE
 60 REM D = DURATION OF NOTE
 70 REM T = OCTAVE OF NOTE
 80 REM C = CHANNEL
 90 ******************************
100 LET V = 6
110 LET C = 1
120 REM ** START OF SCALE **
130 FOR N = 1 TO 8
140 READ P,D,T
150 MUSIC C,T,P,V
160 PLAY C,0,0,0
170 WAIT D
180 PLAY 0,0,0,0
190 NEXT N
200 REM ** END OF SCALE: DATA NEXT **
210 DATA 1,100,3
220 DATA 3,100,3
230 DATA 5,100,3
240 DATA 6,100,3
250 DATA 8,100,3
260 DATA 10,100,3
270 DATA 12,100,3
280 DATA 1,100,4
290 REM ** END OF DATA **
```

Program 21 Scale of C

UNIT 8
Handling numbers

8.1 Introduction

We have demonstrated that computers are not just number crunchers, but in this unit we take a closer look at their arithmetic capacity. We concentrate on fairly straightforward arithmetic, so don't worry about finding it difficult. We're sure that you will be able to cope. We shall continue the 'let's see what happens if' approach, and 'let's get the machine to tell us what is going on'. We hope that you will adopt the same approach with your Oric.

8.2 Averages and the arithmetic mean

'How long have you been taking over each unit of the course so far?' 'Well, on average, about three hours.' If we asked you how long it took you to complete a regular journey, such as going to work, you would answer in a similar manner. Sports enthusiasts use the terms goal average and batting average, atlases abound with pictures of average rainfalls for the months of the year. We talk of average marks in a test or the average age of a group of people, etc.

If we wished to calculate the average or arithmetic mean of the ages of a particular group, we would add up the ages for all of the members of the group and divide that sum by the total number of people in the group. To find the arithmetic mean then involves: adding up, counting, and then dividing the sum by the count.

Example 1
Find the arithmetic mean of the following set of numbers:

6,7,2,5,4,4,9,8.

Solution
Their sum = 6 + 7 + 2 + 5 + 4 + 4 + 9 + 8 = 45.
There are eight numbers.
So their arithmetic mean = 45 / 8 = 5.625

SAQ 1
Find the arithmetic mean of the following set of numbers:

8,4,2,6,1,7,6,1,4.

Arithmetic mean
Example 2
Devise an algorithm and write a program to find the arithmetic mean of the following numbers:

```
200 DATA 56,47,52,65,24,34,59,37,49,66,
210 DATA 38,24,62,76,31,47,66,61,74,45,
220 DATA 66,44,55,67,36,56,54,54,50,43,
230 DATA 18,83,23,79,29,-9999
```

Solution
We will express the algorithm first of all in descriptive form.

1. Start.
2. Set counter to 1.

3. Set sum to 0.
4. Input the next mark.
5. If mark = −9999 then go to 9 otherwise carry on to 6.
6. Add mark to sum.
7. Add 1 to counter.
8. Go to 4.
9. Set total to counter−1.
10. Calculate average = sum/total.
11. Output average.

Figure 1 descriptive algorithm for arithmetic mean

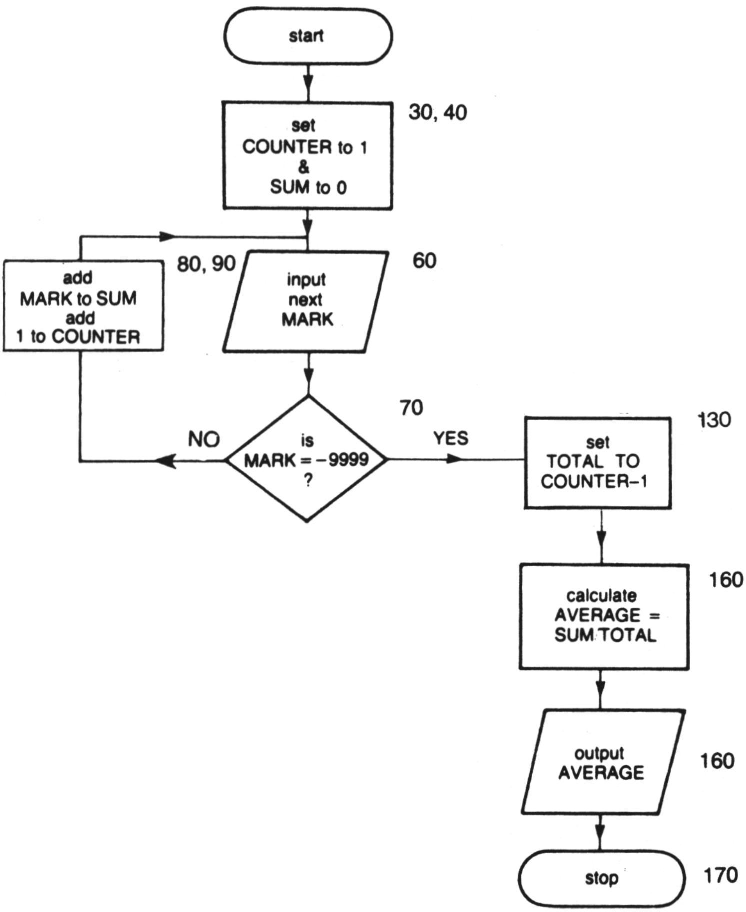


Figure 2 Flowchart for arithmetic mean

```
 10 REM ** Arithmetic Mean **
 15 CLS
 20 PRINT "Arithmetic Mean Program"
 30 C = 1
 40 S = 0
 50 REM ** Processing Loop **
 60 READ M
 70 IF M = -9999 THEN 120
 80 S = S + M
 90 C = C + 1
100 GOTO 50
110 REM **********************
120 REM ** Correct count **
130 N = C - 1
140 REM **********************
150 PRINT
160 PRINT "Average = "; S / N
170 END
180 REM **********************
190 REM ** Data **
200 DATA 56,47,52,65,24,34,59,37,49,66
210 DATA 38,24,62,76,31,47,66,61,74,45
220 DATA 66,44,55,67,36,56,54,54,50,43
230 DATA 18,83,23,79,29,-9999
```

Program 1 Arithmetic mean

```
RUN
Arithmetic Mean Program
Average = 50.571429
```

Ⓚ Program 1.

Please note that many of the programs in this unit require data to be input; this may be done either via an `INPUT` statement, or using `READ` and `DATA`. We have used `INPUT` for some programs, and `READ` with `DATA` for others to give examples of how both options may be used. For Programs 1 and 2 we have used both methods to allow comparison.

```
 10 REM ** Arithmetic Mean **
 15 CLS
 20 PRINT "Arithmetic Mean Program"
 25 PRINT
 30 C = 1
 40 S = 0
 50 PRINT "End list with -9999 "
 55 REM ** Processing Loop **
 60 INPUT "What is the next no.";N
 70 IF N = -9999 THEN 110
 75 REM ** C is counter of inputs **
 80 C = C + 1
 85 REM ** S is sum of numbers input **
 90 S = S + N
100 GOTO 55
110 REM ** Averaging **
120 A = S / C
130 PRINT C;" numbers input"
135 PRINT
140 PRINT "Average = ";A
150 END
```

Program 1a Arithmetic mean (using INPUT)

```
RUN
Arithmetic Mean Program

End list with -9999
What is the next no.? 56
What is the next no.? 47
What is the next no.? 52
What is the next no.? 65
What is the next no.? 24
What is the next no.? 34
What is the next no.? 59
What is the next no.? 37
What is the next no.? 49
What is the next no.? 66
What is the next no.? 38
What is the next no.? 24
What is the next no.? 62
What is the next no.? 76
What is the next no.? 31
What is the next no.? 47
What is the next no.? 66
What is the next no.? 61
What is the next no.? 74
What is the next no.? 45
What is the next no.? 66
What is the next no.? 44
What is the next no.? 55
What is the next no.? 67
What is the next no.? 36
What is the next no.? 56
What is the next no.? 54
What is the next no.? 54
What is the next no.? 50
What is the next no.? 43
What is the next no.? -9999

30 numbers input

Average = 51.2666667
Ready
```

Figure 3 RUN of Program 1a

[K] Program 1a

Exercise 1
Write a program to find the average length of the words in the 'Jude' DATA statements of Example 2 of Unit 5. You can do this by grafting a routine onto Program 1 of Unit 5 which already finds the lengths of words.

Exercise 2
Write a program to find the average score in a simulated experiment of tossing a die 100 times.

Simulation
The expected average score when throwing a die a large number of times is:

$$\frac{1+2+3+4+5+6}{6}=\frac{21}{6}=3.5$$

However, if you look at the answer to Exercise 2, you will see that our run only hit exactly 3.5 once with values ranging from 3.24 to 3.76. That was from the results of

3,000 (30 × 100) throws so you might well ask, 'Is the random number generator biased?' (We did call it 'pseudo' random anyway!) How many experiments do we need to convince us that it is, or is not, biased?

To explore that question fully we need to go into statistical theory that is beyond the scope of this course but we can at least find the mean of these means. All we have to do is take data from the 30 runs of the program:

```
3.56,3.47,3.52,3.65,...
```

and enter them into the `DATA` statements of Program 1 above. This gives us the mean of the means.

You will notice below that we have entered only the decimal parts of the numbers in order to make our data entry a little easier, e.g. 56 instead of 3.56. (We can do this because all the numbers are 3 point something.)

```
56,47,52,65,24,34,59,37,49,66,
38,24,62,76,31,47,66,61,74,45,
66,44,55,67,36,56,54,54,50,43,
```

You will see from the `RUN` of Program 1a, which used this data, that in this case the overall mean of 3,000 throws is 3.51 to two decimal places – a bit more convincing!

Simulation summarised
Simulation is rather a grand word for what we have just done. However, we wanted to emphasise that we can simulate a real life activity without getting deeply involved in statistics. We couldn't toss a die 3,000 times in real life, but with a computer we can collect and process data fairly rapidly.

If your curiosity has been aroused then try the following exercise

Exercise 3
Write a program to find the average score in a simulated experiment of tossing two dice 100 times.

What would you expect the average score to be in this case, and in experiments with three, four ... dice? Are your expectations justified by your experiments?

8.3 Range

While discussing the results of Exercise 2 on the previous page, we quite naturally used the idea of range. We said that the values ranged from 3.24 to 3.76. The process involves finding the lowest and highest values of the set.

Example 3
Devise an algorithm and write a routine to find the maximum and minimum values of the numbers stored in the DATA statements in Example 2. Add this routine to the program to find the arithmetic mean as written as a solution to Example 2.

(If you feel confident enough, treat this Example as an exercise before working through our solution.)

Solution
You may feel we've been here before, in Unit 4, when we found the lowest member

of a list. We could use that approach again but to do so we would first have to put the data in a list form and then sort it twice with the interchange routine. That's a lot of work, so we will look at a shorter approach: trying to find the lowest and highest marks as the data is read in, or input.

We know how to input the data, but what do we do with it as each item is read?

- First we create two stores:
  M for top mark so far
  B for bottom mark so far
- Then we input the first mark (or read it from the data) and put it into both B and M. After all, it's the lowest and highest so far!
- Then we input (or read) each mark and if it is higher than M, put it into M or, if lower than B, put it into B. If neither, just input the next mark.

So a descriptive solution is:

Description
1. Start routine.
2. If counter = 1 then write mark into top and bottom and go to 6, otherwise carry on to 3.
3. If mark > top then write mark into top and go to 6, otherwise carry on to 4.
4. If mark >= bottom then go to 6, otherwise carry on to 5.
5. Write mark into bottom.
6. End routine.

Figure 4

And a flowchart:

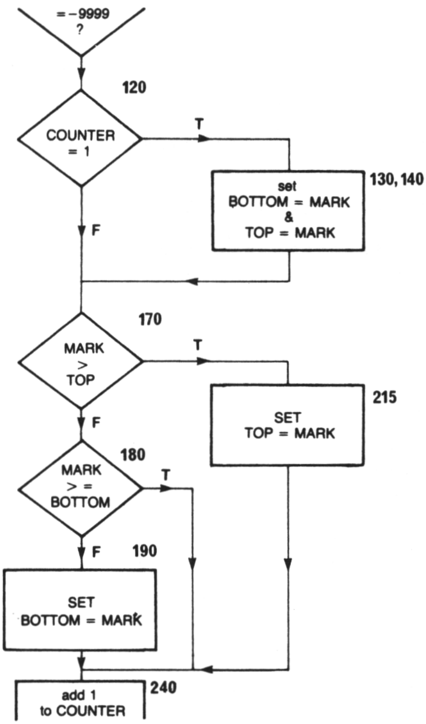


Figure 5

Which do you find easier to follow?

Where there are several branches in a program, the descriptive algorithm can be rather confusing. The two-dimensional display of the flowchart may be more helpful. It's a matter of personal choice; you judge for yourself!

```
 10 REM ** Max and Min **
 15 CLS
 20 PRINT "Mark Handling Program"
 30 REM ** C = Count of Inputs **
 40 C = 1
 50 REM ** S = Sum of inputs **
 60 S = 0
 70 REM ** Input next mark **
 80 INPUT "Next number ";M
100 IF M = -9999 THEN 280
110 IF C > 1 THEN 160
120 REM **********************
130 B = M
140 T = M
150 REM **********************
160 REM ** Main Processing Loop **
170 IF M > T THEN 210
180 IF M >= B THEN 230
185 REM ** New Bottom Mark **
190 B = M
200 GOTO 230
210 REM ** New Top Mark **
215 T = M
220 REM **********************
230 REM ** Increase count **
240 C = C + 1
250 S = S + M
260 GOTO 70
270 REM **********************
280 REM ** Output results **
290 CLS
300 PRINT "Max = ";T
310 PRINT "Min = ";B
320 REM **********************
330 N = C - 1
340 PRINT
350 PRINT "Average = "; S / N
360 END
370 REM **********************
```

the program visits this backwater only the first time around the loop when C=1.

the decisions are made here

Program 2

```
RUN

Mark Handling Program
Next number? 56
Next number? 47
Next number? 52
Next number? 65
Next number? 24
Next number? 34
Next number? 59
Next number? 37
Next number? 49
Next number? 66
Next number? 38
Next number? 24
Next number? 62
Next number? 76
Next number? 31
Next number? 47
```

```
Next number? 66
Next number? 61
Next number? 74
Next number? 45
Next number? 66
Next number? 44
Next number? 55
Next number? 67
Next number? 36
Next number? 56
Next number? 54
Next number? 54
Next number? 50
Next number? 43
Next number? 18
Next number? 83
Next number? 23
Next number? 79
Next number? 29
Next number? -9999

Max = 83
Min = 18
Average = 50.5714286

Ready
```

[K] Program 2.

Here is a READ ... DATA version of Program 2, for comparison.

```
 10 REM ** Max and Min **
 15 CLS
 20 PRINT "Mark Handling Program"
 30 REM ** C = Count of Inputs **
 40 C = 1
 50 REM ** S = Sum of inputs **
 60 S = 0
 70 REM ** Read a value **
 80 READ M
 90 IF M = -9999 THEN 270
110 IF C > 1 THEN 160
120 REM **********************
130 B = M
140 T = M
150 REM **********************
160 REM ** Main Processing Loop **
170 IF M > T THEN 210
180 IF M >= B THEN 230
185 REM ** New bottom mark **
190 B = M
200 GOTO 230
210 REM ** New top mark **
215 T = M
220 REM *********************
230 REM ** Increase count **
240 C = C + 1
250 S = S + M
260 GOTO 70
270 REM *********************
280 REM ** Output results **
290 CLS
300 PRINT "Max = ";T
310 PRINT "Min = ";B
```

```
320 REM *********************
330 N = C - 1
340 PRINT
350 PRINT "Average = "; S / N
360 END
370 REM *********************
380 REM ** Data **
390 DATA 56,47,52,65,24,34,59,37,49,66
400 DATA 38,24,62,76,31,47,66,61,74,45
410 DATA 66,44,55,67,36,56,54,54,50,43
420 DATA 18,83,23,79,29,-9999
```

Program 2a

K Program 2a.

Exercise 4
Write a program to draw up a frequency table for the data in Program 2, using categories:

0-9, 10-19, 20-29, ... 90-99

Suggestion
You could use a score-list

S(0), S(10), S(20), ... S(100)

and for each mark read in, test whether it is less than the top of the second band (10), less than the top of the third band (20), etc. until you find

`MARK` < K true

then increment S(K – 10); this is the approach we have used (see answer).

8.4 Number crunching

We have avoided anything other than fairly simple arithmetic so far in the course, and will continue to do so. But it would be wrong not to give a brief insight to the computer's arithmetic capacity. If your heart sinks at the thought or sight of the following few pages, you will miss no vital programming functions if you pass onto the next section on dry-running and tracing, but we hope you will give it a try. The Oric certainly does take the drudgery out of arithmetic.

Here is a simple program which calculates, for the numbers 1 to 10, their squares (line 50), cubes (line 60) and reciprocals (line 70), and then tabulates the result.

```
 1 REM **Tabulate the squares,
 2 REM cubes and reciprocals
 3 REM for the first ten
 4 REM natural numbers**
 5 CLS
 6 PRINT "      Squares cubes & reciprocals"
 7 PRINT "*******************************"
 8 PRINT
10 PRINT "n         n*n          n*n*n          1/n"
```

```
 20 PRINT
 25 REM ** Processing Loop **
 30 FOR I = 1 TO 10
 35 REM ** Number **
 40 N = I
 45 REM ** Square **
 50 S = I * I
 55 REM ** Cube **
 60 C = I * I * I
 65 REM ** Reciprocal **
 70 R = 1 / I
 75 REM ** Print **
 80 PRINT N;TAB (7);S;TAB (16);C;TAB (28);R
 90 NEXT I
 95 REM ** End of Loop **
100 END
```

Program 3

```
RUN
   Squares cubes & reciprocals
********************************


n    n * n    n * n * n     1 / n

1    1        1             1
2    4        8             0.5
3    9        27            0.333333333
4    16       64            0.25
5    25       125           0.2
6    36       216           0.166666667
7    49       343           0.142857143
8    64       512           0.125
9    81       729           0.111111111
10   100      1000          0.1

Ready
```

🄺 Program 3.

The above program uses TAB. If your Oric doesn't tabulate correctly, try the following variation:

```
  1 REM ** Tabulate the squares,
  2 REM cubes and reciprocals
  3 REM for the first ten natural
  4 REM numbers **
  5 CLS
  6 PRINT "      Squares cubes & reciprocals"
  7 PRINT "*************************************"
  8 PRINT
 10 PRINT "     n     n * n    n * n * n    1 / n"
 20 PRINT
 25 REM ** Processing Loop **
 30 FOR I = 1 TO 10
 35 REM ** Number **
 40 N = I
 42 X = N
 43 GOSUB 200
 44 PRINT N;"    ";
 45 REM ** Square **
 50 S = I * I
 52 X = S
 53 GOSUB 200
 54 PRINT S;"    ";
 55 REM ** Cube **
 60 C = I * I * I
```

```
 62 X = C
 63 GOSUB 200
 64 PRINT C;"    ";
 65 REM ** Reciprocal **
 70 R = 1 / I
 72 IF R < 1 THEN PRINT "0";
 74 PRINT R
 75 ** Print **
 90 NEXT I
 95 REM ** End of Loop **
100 END
200 REM ** Number spacing **
210 IF X <10 THEN PRINT " ";
220 IF X <100 THEN PRINT " ";
230 IF X <1000 THEN PRINT " ";
240 RETURN
```

Program 3 (without TAB)

We have used a subroutine at line 200 to facilitate the spacing. Subroutines are explained in detail in Unit 9.

Raising to a power

We expect that you are familiar with the notation:

$4 \times 4 = 4^2$ (4-squared or 4 raised to the power 2)

$7 \times 7 \times 7 = 7^3$ (7-cubed or 7 raised to the power 3)

$10 \times 10 \times 10 \times 10 \times 10 = 10^5$ (10 raised to the power 5)

In BASIC, raised to the power is shown as ^ or simply ↑.
Any number *N* raised to the power *P* is shown as *N* ^ *P*. is called the exponent and raising to a power is called exponentiation.

Similarly for negative powers:

	N	P	N ^ P
$\frac{1}{4}\times\frac{1}{4}=\frac{1}{4\times4}=4^{-2}$	4	−2	4 ^ (−2)
$\frac{1}{7}\times\frac{1}{7}\times\frac{1}{7}=\frac{1}{7\times7\times7}=7^{-3}$	7	−3	7 ^ (−3)
$\frac{1}{10}\times\frac{1}{10}\times\frac{1}{10}\times\frac{1}{10}\times\frac{1}{10}\times\frac{1}{10}=\frac{1}{10\times10\times10\times10\times10}=10^{-5}$	10	−5	10 ^ (−5)

Fractional powers (positive and negative) are also possible but we will not be concerned with them.

We can use the ^ notation instead of * in Program 3. Thus Program 3 re-written with ^ becomes:

```
1 REM ** Tabulate the squares,
2 REM cubes and reciprocals
3 REM for the first ten natural
4 REM numbers **
5 CLS
6 PRINT "      Squares cubes and reciprocals"
```

```
  7 PRINT "****************************************"
  8 PRINT
 10 PRINT "    n      n ^ 2      n ^ 3      1/n"
 20 PRINT
 25 REM ** Processing Loop **
 30 FOR I = 1 TO 10
 35 REM ** Number **
 40 N = I
 42 X = N
 43 GOSUB 200
 44 PRINT N;"    ";
 45 REM ** Square **
 50 S = INT(1 ^ 2)
 52 X = S
 53 GOSUB 200
 54 PRINT S;"    ";
 55 REM ** Cube **
 60 C = INT(1 ^ 3)
 62 X = C
 63 GOSUB 200
 64 PRINT C;"    ";
 65 REM ** Reciprocal **
 70 R = I ^ (-1)
 74 PRINT R
 90 NEXT I
100 END
200 REM ** Number spacing **
210 IF X <10 THEN PRINT " ";
220 IF X <100 THEN PRINT " ";
230 IF X <1000 THEN PRINT " ";
240 RETURN
```

Program 4 (without TAB)

Lines 50 and 60:
the exponentiation function does not give an exact answer on the Oric, so we need to round off. The INT function is covered in Section 8.7 in detail.

Note need for INT as ^ is not exact!

```
RUN

          Squares cubes & reciprocals
     *************************************
     n       n^2        n^3        1 / n
     1        1          1         1
     2        4          8         .5
     3        9         27         .333333333
     4       16         64         .25
     5       25        125         .2
     6       36        216         .166666667
     7       49        343         .142857143
     8       64        512         .125
     9       81        729         .111111111
     10     100       1000         .1
```

Sequences and their sums

Calculating the individual terms in a sequence, or the sum of the first *N* terms, is a very great labour without a computer. How long would it take you to evaluate the terms of

$$\frac{1}{1^2}, \frac{1}{2^2}, \frac{1}{3^2}, \ldots \frac{1}{N^2} \ ?$$

Well, it's very easy with Program 5.

```
 5 CLS
10 PRINT "Series Calculation"
20 PRINT
30 PRINT "n   n ^ (-2)"
40 REM ** Start of processing **
50 FOR N = 1 TO 10
60 PRINT N,N ^ (-2)
70 NEXT N
80 END
```

Program 5 (with TAB)

```
 5 CLS
10 PRINT "Series Calculation"
20 PRINT
30 PRINT"n     n ^ (-2)
40 REM ** Start of processing **
50 FOR N = 1 TO 10
60 PRINT N;
62 IF N < 10 THEN PRINT"  ";
64 PRINT "   ";N ^ (-2)
70 NEXT N
80 END
```

Program 5 (without TAB)

```
RUN
Series Calculation

n             n ^ (-2)
1             1
2             0.25
3             0.111111111
4             .0625
5             .04
6             .0277777778
7             .0204081633
8             .015625
9             .012345679
10            .01

Ready
```

🄺 Program 5.

Exercise 5
Write a program to find how many terms of the series

$$1,\ \frac{1}{2},\ \frac{1}{3},\ \frac{1}{4},\ \ldots$$

are needed to make their sum exceed 2.4.

Exercise 6
Modify the program from Exercise 5 to find out how many terms are needed for the sum

$$\text{sum} = 1+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\ldots$$

to exceed 1.5.

Exercise 7
Factorials are interesting numbers. Factorial 4 = $4 \times 3 \times 2 \times 1$ and is usually written 4! So

$$\text{Factorial } 7 = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 7!$$

$$\text{Factorial } N = N \times (N - 1) \times (N - 2) \times \ldots \times 1 = N!$$

Write a program to evaluate the factorial of any positive integer.

Exercise 8
We wrote a rather clumsy program to evaluate the yield (Y) on a deposit (D) at a compound interest percentage (P) way back in Unit 1. A neater formula is

$$Y = D \times (1 + P/100)^T$$

where T is the number of years of the investment.

Write a program to evaluate the yield, using this formula, for various deposits, percentages and time periods.

8.5 Dry running

Often we find that a program either does not work at all, or not to our complete satisfaction. If we have reasonable access to a computer we may sit at the keyboard until we trace the fault, but when all else fails we may be forced to sit down with pencil and paper and think hard. Stepping through an algorithm line by line with pencil and paper is called dry running.

We shall illustrate dry running by looking at Program 17 which was the solution to Exercise 5.

```
 10 REM ** Sum of reciprocals **
 15 CLS
 20 PRINT "How many terms in the reciprocal"
 25 PRINT "series are needed for the sum to"
 27 PRINT "exceed 2.4?"
 30 LET S = 0
 35 LET N = 1
 40 REM ** Start of processing loop **
 50 S = S + 1 / N
 60 IF S > 2.4 THEN 90
 70 N = N + 1
 80 GOTO 40
 90 REM ** OUTPUT **
100 PRINT
110 PRINT "Sum = ";S;" the no. of terms is ";N
120 END
```

Program 17 (from Exercise 5)

```
RUN
How many terms in the reciprocal
series are needed, for the sum to
exceed 2.4?

Sum = 2.45  the no. of terms is 6
Ready
```

Tracing means finding and recording each step, the line number executed at that step and the values of the variables after that line has been executed. So, for Program 18 we need the headings:

step No.	line No.	N	S
1			
2			
etc			

We omit from the trace lines which don't affect the variables, i.e.:

```
REM      (lines 10,40 & 90)
GOTO     (line 80)
PRINT    (lines 20,25,27,100,110)
```

Apart from these, the program steps are as follows:

step no.	line no.	n	s	
1	30	0	0	
2	35	1	0	
3	50	1	1	
4	60	1	1	Note effect of GOTO 40 line 60
5	70	2	1	
6	50	2	1.5	
7	60	2	1.5	
8	70	3	1.5	GOTO again
9	50	3	1.83	
10	60	3	1.83	
11	70	4	1.83	GOTO
12	50	4	2.08	
13	60	4	2.08	
14	70	5	2.08	GOTO
15	50	5	2.28	
16	60	5	2.28	
17	70	6	2.28	GOTO
18	50	6	2.45	

Figure 6

Tracing
Some BASIC interpreters provide a TRACE command

TRON turns the trace on

e.g. at the start of a program

```
10 TRON
```

If you only want to check part of your coding, it is possible to switch the trace off again:

TROFF is the required keyword

e.g. `90 TROFF`

The trace prints out the numbers of lines executed when the program is `RUN`. The numbers are enclosed in square brakcets, to avoid confusion with the normal screen outputs. Following through these numbers enables you to follow the path the computer has taken through your program. We shall look at tracing later in this unit.

SAQ 2
Complete a dry-run on Program 6 starting when line 20 has just been executed and continuing until the condition in line 60 is true. Draw up the table with the same headings as in the above example.

```
 10 REM ** Sum of squares **
 15 CLS
 20 S = 0
 30 N = 1
 40 REM ** Add next term **
 50 S = S + INT(N ^ 2)
 60 IF S > 50 THEN 90
 70 N = N + 1
 80 GOTO 40
 90 REM ** Output **
100 PRINT "Sum of squares"
110 PRINT
120 PRINT "Sum = ";S;" no. of terms is ";N
130 END
```

Program 6

8.6 REPEAT ... UNTIL ...

We have just used two programs which were written to answer numeric questions: the 'Sum of reciprocals' program on page 212 (Program 17 from Exercise 5) answered the question 'How many items in the reciprocal series are needed for the sum to exceed 2.4?'. The 'Sum of squares' program above (Program 6) answered the question 'how many terms are needed in the squares series for the sum to exceed 50?'.

If you look at these programs you will see that they both have a loop at their centres. For the reciprocals program we see

```
40 REM **Start of processing loop**
50 S = S > 2.4 THEN 90
70 N = N + 1
80 GOTO 40
```

Line 80 will keep repeating the loop until the condition in line 60 is true.

For the squares program we have

```
40 REM **Add next term**
50 S = S + INT(N ∧ 2)
60 IF S > 50 THEN 90
70 N = N + 1
80 GOTO 40
```

Again, here the structure is very similar.

We had to use a simple `IF ... THEN ...` loop as we did not know in advance the number of times the loop would need to be repeated. This stopped us from using the prefered `FOR ... NEXT` loop. There is a way in Oric BASIC to overcome this: `REPEAT ... UNTIL ...` loop.

Let us look at altered versions of the reciprocals and squares programs:

```
 10 REM **Sum of reciprocals**
 15 CLS
 20 PRINT "How many terms in the reciprocal"
 25 PRINT "series are needed, for the sum to"
 27 PRINT "exceed 2.4?"
 30 S = 0
 35 N = 0
 40 REM **Start of processing loop**
 50 REPEAT
 60 :   N = N + 1
 70 :   S = S + 1 /N
 80 UNTIL S > 2.4
 90 REM **OUTPUT**
100 PRINT
110 PRINT "Sum = ";S;" the no. of terms is ";N
120 end
```

our 'new' type of loop

Program 18 (From Exercise 5) with ***REPEAT ... UNTIL ...*** option

```
 10 REM **Sum of squares**
 15 CLS
 20 S = 0
 30 N = 0
 40 REPEAT
 50 N = N + 1
 60 S = S + INT(N ^ 2)
 70 UNTIL S > 50
 90 REM **Output**
100 PRINT "Sum of squares"
110 PRINT
120 PRINT "Sum = ";S;" no. of terms is ";N
130 END
```

Again the 'new' type of loop

Program 6 (modified) with ***REPEAT ... UNTIL ...*** loop

The first statement in the loop is `REPEAT`. All statements between `REPEAT` and the final line of the loop which begins `UNTIL ...` are carried out over and over again until the condition in the `UNTIL ...` line is true. This gets rid of the `IF ... THEN ...` line number and the `GOTO` line number statements which spoil our structured approach to programming. The advantage to us when 'number crunching' is that the `REPEAT ... UNTIL ...` loop enables us to work to a given level of accuracy in our answers.

8.7 The representation of numbers

So far in the course we have left in abeyance a number of questions about the representation of numbers in our programs. We do not intend to consider the mathematics of number representation in computers in general, but we need to tidy up our ideas about numbers.

In general terms computers must be able to process and store the following types of number:

(a) positive and negative whole numbers (integers or counting numbers)
(b) fractions and numbers which are partly whole and partly fractional (measuring numbers);
(c) very large and very small numbers;
(d) the number zero.

The single most important piece of advice that we can give you at this stage is that if a number has been involved in any sort of calculation within a program then consider it with a certain amount of suspicion. The reason for this statement is that numbers within a computer are stored and manipulated in binary form, that is to base 2, rather than to the base 10 with which we are familiar. In our familiar decimal notation you will recall that a number like 1/3 or 1/7 is incapable of exact expression, e.g. 1/3 = 0.3333... We assume that by adding on as many 3's to this number as we wish. we can achieve an acceptable level of accuracy in any particular problem. In the binary system, in just the same way, some numbers cannot be expressed exactly, e.g. in decimal form we can say that 1/10 = 0.1 exactly, but when this number is changed into binary form it cannot be represented exactly.

Most BASIC interpreters allow for numbers to be expressed to an accuracy of six decimal digits. In decimal form, then, the number 3 1/10 could be represented as 3.100000 to six decimal digit accuracy. However, if this number had been the result of some calculation within a computer, we might find that the number output was 3.09999 or 3.10001. So, in general, you must always be suspicious of the least significant digit in any answer (i.e. the digit on the far right of the number).

Program 7 shows how this type of inaccuracy may occur. The `FOR ... NEXT` loop adds 4.0, 4.1 and 4.25 into locations S, T, and U one thousand times. Now 4.0 and 4.25 are exactly represented in binary form, but 4.1 is not. We can see that the result of this repetitious summation is exact in respect of 4.0 and 4.25, but not for 4.1 where the error is 0.00016 in 4100.00. Obviously, we would avoid writing programs involving such repetitions wherever possible, but we hope that the program will act as a warning about possible inaccuracies.

You have already seen that the exponentiation function (raising to the power of) is not exact even for whole number inputs in Oric BASIC.

```
 10 REM **Accuracy demonstration**
 15 CLS
 20 S = 0
 30 T = 0
 40 U = 0
 50 FOR I = 1 TO 1000
 60 S = S + 4.0
 70 T = T + 4.1
 80 U = U + 4.25
 90 NEXT I
100 PRINT S
```

```
110 PRINT T
120 PRINT U
130 END
```

Program 7

```
RUN
4000
4100.00016
4250

Ready
```

[K] Program 7.

Small and large numbers

The six to ten decimal digits in microcomputers do not allow us to cope with very small or very large numbers. So these numbers are represented in BASIC in exponential form.

Small numbers

0.000586321 means

$$\frac{586321}{1,000,000,000} = \frac{586321}{10^9}$$

which could be expressed as 586321×10^{-9}. We could also express 0.000586321 as $\frac{0.586321}{1000} = 0.586321 \times 10^{-3}$.

In BASIC this last number would be written as 0.586321E − 3.

Similarly,
$0.0234539 = 2.34539 \times 10^{-2}$ = 2.34539E − 2
and
$0.00000000959734 = 0.959734 \times 10^{-8}$ = 0 .9 . . . E − 8

E stands for exponent and the base for exponentiation is 10. So E−4 means move the decimal point point four places to the left, and E+9 means move the decimal point nine places to the right.

Large numbers

$12368500 = 1.23685 \times 10^7$ = 1.23685E + 7
$935.432 = 0.935432 \times 10^3$ = 0.935432E + 3
959734000000000000000 = 0.959734E + 21

Most BASIC interpreters have a range of at least E − 32 to E + 32.

[K] Find out how the Oric represents small and large numbers with this program.

```
10 REM **Number demonstration**
15 CLS
20 PRINT "Number", "Representation"
30 FOR I = -10 TO 10
40 PRINT "10^";I,10 ^ I
50 NEXT I
60 END
```

Program 8

```
10 REM **Number demonstration**
15 CLS
20 PRINT "Number","Representation"
30 FOR I = -10 TO 10
40 PRINT "10 ^ ";I;
41 PRINT " ";
42 IF I < 10 THEN PRINT " ";
43 IF I < 0 THEN 46
44 PRINT " ";
46 IF I <-9 THEN 48
47 PRINT " ";
48 PRINT 10 ^ I
50 NEXT I
60 END
```

Version of Program 8, avoiding tabulation.

```
RUN

Number          Representation
10^-10    9.99998E - 11
10^-9     1E-09
10^-8     1E-08
10^-7     1E-07
10^-6     1E-06
10^-5     1E-05
10^-4     1E-04
10^-3     1E-03
10^-2     .01
10^-1     .1
10^0      1
10^1      10
10^2      100
10^3      1000
10^4      10000
10^5      100000
10^6      1000000
10^7      10000000
10^8      100000000
10^9      1E+09
10^10     1E+10

Ready
```

🄺 Program 8.

8.8 The INT-function for rounding

For some problems we need to round off the result of a calculation to the nearest whole number.

e.g.6.6 to the nearest whole number is 7.
    7.4 to the nearest whole number is 7.

This is especially true if we are not confident about the last figure accuracy of a decimal number,
e.g. 6.99999 or 7.00001 for 7.

The function `INT(X + 0.5)` does this rounding for us, as the following program demonstrates.

```
10 REM **INT for rounding**
15 CLS
20 PRINT "X";TAB(10);"INT(X)";TAB(20);"INT(X + 0.5)"
30 FOR I = -1.4 TO -2.6 STEP -0.1
40 PRINT I;TAB(10);INT(I);TAB(20);INT(I + 0.5)
50 NEXT I
60 END
```

Program 9a (`TAB` version)

```
10 REM **INT for rounding**
15 CLS
20 PRINT "X        INT(X)      INT(X + 0.5)"
30 FOR I = -1.4 TO -2.6 STEP -0.1
40 PRINT I;
41 IF I = -2 THEN PRINT "  ";
42 PRINT "      ";INT(I);
44 PRINT "       "INT(I + 0.5)
50 NEXT I
60 END
```

Program 9a (without `TAB`)

```
RUN

X         INT(X)    INT(X + 0.5)
-1.4       -2       -1
-1.5       -2       -1
-1.6       -2       -2
-1.7       -2       -2
-1.8       -2       -2
-1.9       -2       -2
-2         -2       -2
-2.1       -3       -2
-2.2       -3       -2
-2.3       -3       -2
-2.4       -3       -2
-2.5       -3       -2
-2.6       -3       -3
Ready
```

Program 9a

Changing line 30 for `30 FOR I = 1.4 TO 2.6 STEP(.1)` gives

```
X         INT(X)    INT(X + 0.5)
1.4       1         1
1.5       1         2
1.6       1         2
1.7       1         2
1.8       1         2
1.9       1         2
2         2         2
2.1       2         2
2.2       2         2
2.3       2         2
2.4       2         2
2.5       2         2
2.6       2         3
```

Program 9b

[K] Program 9a.

SAQ 3

What print-outs will we get from the above program if we change line 30 as follows:

(a) `30 FOR I=.4 TO 2.2 STEP (.2)?`

(b) `30 FOR I=.4 TO -.6 STEP (-.2)?`

8.9 The ABS-function

An arithmetic function related to the above ideas is to find the modulus or absolute value of a number. It sounds rather grand, but is very simple.

ABS(X) simply gives us the positive value of X.
e.g. ABS(23) = 23, ABS(−23) = 23

The following program illustrates the function.

```
 10 REM **The ABS function**
 15 CLS
 20 PRINT "Using the ABS function"
 30 PRINT
 45 PRINT
 50 REM **Start of loop**
 60 FOR I = 1 TO 4
 70 INPUT "X =";X
 80 INPUT "Y =";Y
 90 PRINT"X + Y = ";X + Y
100 PRINT"ABS(X + Y) = ";ABS(X + Y)
110 PRINT
130 NEXT I
140 END
```

Program 10

```
RUN
Using the ABS function

X =?  5
Y =?  7
X + Y = 12
ABS(X + Y) = 12

X =?  5
Y =?  -7
X + Y = -2
ABS(X + Y) = 2

X =?  -5
Y =?  7
X + Y = 2
ABS(X + Y) = 2

X =?  -5
Y =?  -7
X + Y = -12
ABS(X + Y) = 12

Ready
```

[K] Program 10.

SAQ 4
What will the print-out be if we input
9,14,11,−2,−4,13,−7,−8?

8.10 Iteration

The process of making a guess at a value, testing it, making a better guess and testing again, etc., until we home in on an item or value is known as iteration. The essence of iteration involves:

- making an arbitrary start
- guessing how accurate this point is
- refining this in a sequence of repeated operations.

Square root by iteration
BASIC can do square roots directly, as this program demonstrates:

```
10 REM **Direct square roots**
15 CLS
20 FOR X = 33 TO 63 STEP 10
30 PRINT X;"    "X ^ 0.5;"    ";SQR(X)
40 NEXT X
50 END
```

Program 11

(`SQR(X)` gives the square root of X provided that X > = 0.)

```
RUN
33          5.74456265       5.74456265
43          6.55743853       6.55743853
53          7.28010989       7.28010989
63          7.93725393       7.93725393
Ready
```

However, we shall also show you how to find a square root by an iterative method; not because that's how you would normally do it, but because it's an easy example through which to demonstrate iteration.

The method
If you want to square root a number N then

- take a guess at the square root, say G
- work out N/G
- then the average of G and N/G is an even better guess than your first one, i.e. it is closer to the unknown square root than your first guess
- go back to the beginning using this new 'better' guess.

We shan't prove this here (it's in plenty of maths textbooks) but we shall show its working in a simple case.

To square root	$N = 12$
Guess	$G = 2$
Work out	$N / G = 6$
Take average	$\dfrac{G + N / G}{2} = \dfrac{2 + 6}{2} = 4$

So 4 is the new guess:

Guess $G = 4$

$$N / G = 3$$

$$\frac{G + N / G}{2} = 3.5$$

So 3.5 is the new guess.

In table form this looks like:

G	N / G	$\frac{G + N / G}{2}$
2	12 / 2 = 6	(2 + 6) / 2 = 4
4	12 / 4 = 3	(4 + 3) / 2 = 3.5
3.5	12 / 3.5 = ...	...

Figure 7

SAQ 5

To make sure that you have grasped the process, draw up a table similar to that above, but make your first guess 1.

Iteration ... stop

The question now is: 'How do we stop the process?' Well, we want to stop the process when the square of our guess becomes as close as possible to N. What do we mean by 'as close as possible?' The answer is that it's up to you. You're in charge! How accurate do you want the square root to be? If, for example, we wish to find the square root of 12 to two decimal places, then the difference between G * G and N would have to be

```
<0.005.
```

We don't need to know whether G * G is bigger or smaller than N – only the difference matters. So we are back to `ABS`. If

```
ABS(N - G * G) <0.005
```

then stop is what we are after.

Descriptive algorithm for square root iteration

1. Start.
2. Input the number whose square root is sought.
3. Input the accuracy required.
4. Input a guess at the square root.
5. If the guess is within the accuracy required then go to 8, otherwise continue with the next statement.
6. Make the guess more accurate.
7. Return to 5.
8. Output the square root value found.
9. Stop.

Figure 8

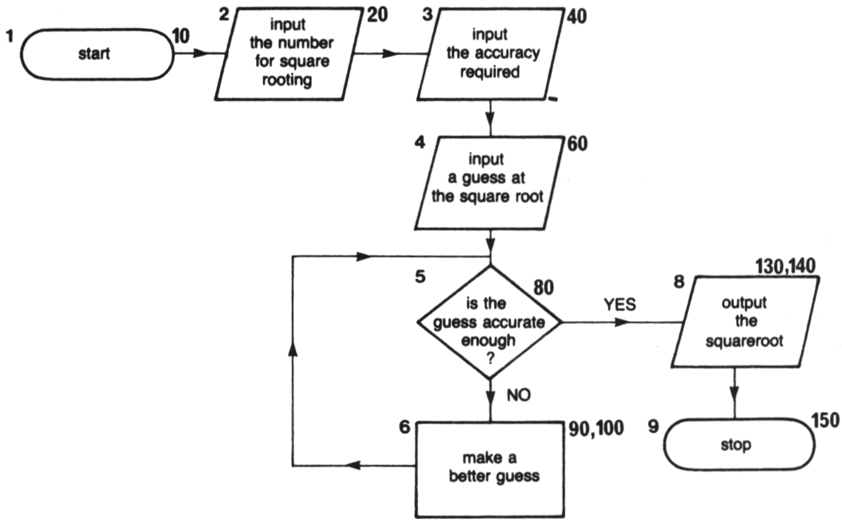


Figure 9 Flowchart for square root iteration

```
 10 REM **Square root- iteration**
 20 INPUT "No. for square-rooting?";N
 30 PRINT
 40 INPUT "Accuracy required";A
 50 PRINT
 60 INPUT "Your guess";G
 70 REM **start of iteration**
 80 REPEAT
 90 G = 0.5 * (G + (N / G))
100 UNTIL ABS(N - G * G) < A
110 REM **Output**
120 PRINT
130 PRINT "The square root of ";N
140 PRINT "is ";G
150 END
```

Program 12 Square root iteration

```
RUN
No. for square-rooting? 12
Accuracy required? 5E -03
Your guess? 2

The square root of 12
is 3.46428572
Ready
```

(shown as 0.005 on screen)

```
RUN
No. for square-rooting? 12
Accuracy required? 5E -04
Your guess? 2

The square root of 12
is 3.46410162

Ready
```

(shown as 0.0005 on screen)

🄺 Program 12.

Tracing the iterative process

Now the result of Program 12 is not very startling. As we pointed out earlier on, we can find N directly with a computer. But we wanted a simple example to demonstrate interation at work so we will now take a closer look at what is happening. All we need to do in Oric BASIC is to add the line:

```
5 TRON
```

at the start of the program.

Now look at the run of the program:

```
RUN
[10][20] No. for square-rooting? 12
[30]
[40] Accuracy required? .005
[50]
[60] Your guess? 2
[70] [80] [90] [100] [90] [100] [90] [100] [110] [120]
[130] The square root of 12
[140] is 3.46428572
[150]
Ready
```

once around the loop

loops can be seen here

Lines 90 and 100 are the `REPEAT ... UNTIL ...` loop, and have been repeated three times. It took three iterations to get an accuracy of 0.005.

Now let us try another run, with an accuracy of .0005, but with the data otherwise unchanged.

```
[10][20] No. for square-rooting? 12
[30]
[40] Accuracy required? .0005
[50]
[60] Your guess? 2
[70][80] [90] [100] [90] [100] [90] [100] [90] [100] [110] [120]
[130] The square root of 12
[140] is 3.46410162
[150]

Ready
```

loops here again

The REPEAT ... UNTIL ... loop was used four times. Increasing the accuracy tenfold still only takes four loops.

If we give a negative number:

```
RUN
10 20 No. for square-rooting? -4
30
40 Accuracy required .005
50
60 Your guess is? 2
70  80  90  100  90

? DIVISION BY ZERO ERROR IN 90

Ready
```

Depending on the size of the number, the computer will come up with the message shown above, or will output gibberish, if we input a negative number. We can break into the program with CTRL and C, if necessary.

Exercise 9
Change the square root program above to produce cube roots. If G is a guess at the cube root of N then $\frac{1}{2}(G + N / G^2)$ will be a better guess.

Assignment 8

1. Produce a conversion chart for changing litres to gallons, and vice versa, displayed on the screen. Make the litre answers rounded to the nearest whole number, and the gallon answers rounded to two places of decimals; use the conversion 1 gallon equals 4.544 litres.

   Notes: set the numbers 1 to 15 down the centre of the screen; output the litre equivalents down the left-hand side, and the gallon equivalents down the right-hand side; e.g. the first row would have 5 under the litres column, 1 in the centre column, and 0.22 in the gallons column, because 1 gallon is 5 litres, and 1 litre is 0.22 gallons. Produce 15 rows of the table, which will leave space for titles, etc.

2. Produce a program to calculate and display your car's petrol consumption, from the distance travelled since the fuel tank was last filled up, and the amount of fuel needed to refill the tank.

   Hint: you should use the INT(X + 0.5) function to provide reasonable levels of accuracy.

   If you were feeling very confident, you could allow your fuel consumption program to cater for either miles/gallon or km/litre! (1 mile = 1.609 km; 1 gallon = 4.544 litres)

Objectives of Unit 8

- Calculate (manually) an arithmetic mean ☐
- Write a program to calculate arithmetic means ☐
- Write a program to find the largest and smallest item in a data list ☐
- Use *, / and ^ in programs ☐
- Dry run a program ☐
- Use `REPEAT ... UNTIL ...` ☐
- Interpret numbers in `E` notation ☐
- Use `INT(X + 0.5)` for rounding ☐
- Use `ABS(X)` ☐
- Write programs for iterative routines ☐
- Insert trace print lines in a program ☐
- Use trace (`TRON` and `TROFF`) ☐

Answers to SAQs and Exercises

SAQ 1

Sum = 8 + 4 + 2 + 6 + 1 + 7 + 6 + 1 + 4 = 39
There are nine numbers.
So the arithmetic mean is 39 / 9 = 4.333...

Exercise 1

```
 10 REM **Average Length**
 15 CLS
 20 PRINT "Average length program"
 30 S = 0
 40 C = 1
 50 REM **READ loop**
 60 READ W$
 70 IF W$ = "ZZZZ" THEN 180
 80 L = LEN (W$)
 90 S = S + L
100 C = C + 1
110 GOTO 50
120 REM **********************
130 DATA The,horse,stood,still,till,he,had,finished,the,hymn
140 DATA which,Jude,repeated,under,the,sway,of,a,polytheistic
150 DATA fancy,that,he,would,never,have,thought,of,humouring
160 DATA in,broad,daylight,ZZZZ
170 REM **********************
180 N = C - 1
190 A = S / N
200 PRINT "Average length of the words"
205 PRINT "is ";A
210 END
```

Program 13

```
RUN

Average length of the words is 4.64516129

Ready
```

[K] Program 13.

Exercise 2

```
 10 REM **Mean of 100 throws**
 20 REM **of one die**
 25 CLS
 30 PRINT "Throw one die 100 times"
 40 S = 0
 50 REM **start of loop**
 60 FOR I = 1 TO 100
 70 LET X = INT (RND(1) * 6) + 1
 80 S = S + X
 90 NEXT I
100 REM **Output**
110 PRINT
120 PRINT "Average Score = ";S / 100
130 END
```

Program 14

```
RUN

Throw one die 100 times
Average Score = 3.71

Ready
```

[K] Program 14.

Exercise 3

```
 10 REM **Mean of 100 throws**
 20 REM **of two dice**
 25 CLS
 30 PRINT "Throw two dice 100 times"
 40 S = 0
 50 REM **start of loop**
 60 FOR I = 1 TO 100
 70 X = INT (RND(1) * 6) + 1
 75 Y = INT (RND(1) * 6) + 1
 80 S = S + X + Y
 90 NEXT I
100 REM **Output**
110 PRINT
120 PRINT "Average Score = ";S / 100
130 END

RUN

Throw two dice 100 times
Average Score = 6.67

Ready
```

Program 15

[K] Program 15.

Exercise 4

```
 10 REM **Frequency Table/Histogram**
 15 CLS
 20 PRINT "......Histogram......"
 30 DIM S(101)
 40 REM **********************
 45 PRINT
 46 PRINT "Data...(Range 0-100)"
 48 REM **Input loop**
 50 INPUT M
 51 IF M = -9999 THEN 145
 80 K = 11
 85 REM **Find range of data item**
 90 IF M < K THEN 120
100 K = K + 10
110 GOTO 85
120 REM **Store data in table**
125 S(K - 10) = S(K - 10) + 1
130 GOTO 48
140 REM **Output table**
145 PRINT
147 PRINT "Table..."
148 PRINT
150 FOR K = 1 TO 101 STEP 10
160 : K - 1;"-";TAB 7;S(K);TAB 12;
163 REM **Histogram loop**
165 FOR P = 1 TO S(K)
166 PRINT "*";
167 NEXT P
168 REM **Histogram complete**
170 PRINT
180 NEXT K
190 END
```

K for score category or class interval

correct category found. Increment counter for that category

Program 16 with TAB

If your Oric's TAB function is not working correctly, use the version of Program 16 shown next.

```
 10 REM **Frequency Table/Histogram**
 15 CLS
 20 PRINT "......Histogram......"
 30 DIM S(101)
 40 REM ******************************
 45 PRINT
 46 PRINT "Data...(Range 0-109)
 48 REM **Input Loop**
 50 INPUT M
 51 IF M = -9999 THEN 145
 80 K = 11
 85 REM **Find range of data item**
 90 IF M < K THEN 120
100 K = K + 10
110 GOTO 85
120 REM **Store data in table**
125 S(K - 10) = S(K - 10) + 1
130 GOTO 48
140 REM **Output table**
145 PRINT
147 PRINT "Table..."
```

```
148 PRINT
150 FOR K = 1 TO 101 STEP 10
152 IF K = 1 < 10 THEN PRINT "  ";
154 IF K - 1 < 100 THEN PRINT " ";
156 PRINT K - 1;"-    ";
158 PRINT S(K);"      ";
163 REM **Histogram loop**
165 FOR P = 1 TO S(K)
166 PRINT "*";
167 NEXT P
168 REM **Histogram complete**
170 PRINT
180 NEXT K
190 END
```

Program 16 without TAB

[K] Program 16.

```
RUN

......Histogram......

Data...(Range 0-100)
? 56
? 47
? 52
? 65
? 24
? 34
? 59
? 37
? 49
? 66
? 38
? 24
? 62
? 76
? 31
? 47
? 66
? 61
? 74
? 45
? 66
? 44
? 55
? 67
? 36
? 56
? 54
? 54
? 50
? 43
? 18
? 83
? 23
? 79
? 29
? -9999

Table ...
```

```
0          0    *
10  -      1    *
20  -      4    ****
30  -      5    *****
40  -      7    *******
50  -      7    *******
60  -      7    *******
70  -      3    ***
80  -      1    *
90  -      0    *
100 -      0    *
Ready
```

Exercise 5

```
 10 REM **Sum of reciprocals**
 15 CLS
 20 PRINT "How many terms in the reciprocal"
 25 PRINT "series are needed for the sum to"
 27 PRINT "exceed 2.4?"
 30 S = 0
 35 N = 0
 40 REM **Start of processing loop**
 50 REPEAT
 60 :   N = N + 1
 70 :   S = S + 1 / N
 80 UNTIL S > 2.4
 90 REM **Output**
100 PRINT
110 PRINT "Sum = ";S;" the no. of terms is ";N
120 END
```

Program 17 REPEAT ... UNTIL ...

```
RUN
How many terms in the reciprocal
series are needed for the sum
to exceed 2.4?

Sum = 2.45   the no. of terms is 6
Ready
```

[K] Program 17.

Exercise 6

```
 10 REM **Sum of N ^ (-2)**
 15 CLS
 20 PRINT "How many terms in the series"
 25 PRINT "N ^ (-2) are needed for the sum to"
 27 PRINT "exceed a given value?"
 30 S = 0
 35 N = 1
 40 REM **Start of processing loop**
 50 REPEAT
 60 :   N = N + 1
 70 :   S = S + N ^ (-2)
 80 UNTIL S > 1.5
 90 REM **Output**
100 PRINT
110 PRINT "Sum = ";S;" the no. of terms is ";N
120 END
```

Program 18

```
RUN
How many terms in the series
N ^ (-2) are needed for the sum to
exceed a given value?

Sum = 1.51179705 the no. of terms is 7
Ready
```

You can, of course, change line 80 to explore the number of terms needed for the sum to exceed other values, e.g.

```
RUN

How many terms in the series
N ^ (-2) are needed for the sum to
exceed a given value?

Sum = 1.60049693   the no. of terms is 22
```

for 1.6

```
Ready
```

```
RUN

How many terms in the series
N ^ (-2) are needed for the sum to
exceed a given value?

Sum = 1.61103901   the no. of terms is 29
```

for 1.61

```
Ready
```

```
RUN

How many terms in the series
N ^ (-2) are needed for the sum to
exceed a given value?

Sum = 1.62024396   the no. of terms is 40
```

for 1.62

```
Ready
```

```
RUN

How many terms in the series
N ^ (-2) are needed for the sum to
exceed a given value?

Sum = 1.63011952   the no. of terms is 67
```

for 1.63

```
Ready
```

What a good starting point these and similar programs would make to lessons about limits and convergence of number series!

🄺 Program 18.

Exercise 7

```
 10 REM **Factorials**
 15 CLS
 20 PRINT "Evaluate the factorial of any"
 25 PRINT "positive integer."
 30 PRINT
 40 PRINT "To end the run, enter -9999"
 50 PRINT
 55 REM **Input Loop**
 60 PRINT "Next factorial";N
 70 IF N = -9999 THEN 150
 80 F = 1
 90 FOR I = 1 TO N
100 F = F * I
110 NEXT I
120 PRINT N,F
130 PRINT
140 GOTO 55
150 END
```

Program 19

```
RUN
Evaluate the factorial of any
positive integer

To end the run, enter -9999

Next factorial?  1
1    1

Next factorial?  3
3    6

Next factorial?  5
5    120

Next factorial?  7
7    5040

Next factorial?  9
9    362880

Next factorial?  11
11          39916800

Next factorial?  13
13          6.2270208E+09 ———— what's happened here?
                                    see the section on number
                                    representation.

Next factorial?  -9999

Ready
```

🄺 Program 19.

Exercise 8

```
 10 REM **Compound Interest**
 15 CLS
 20 PRINT "Compound Interest Calculations"
 30 PRINT
 40 INPUT "What is your deposit (£)";D
 50 PRINT
 60 INPUT "What rate of interest (%)";P
 70 PRINT
 80 INPUT "For how many years";T
 90 PRINT
100 PRINT "Time (years)  Yield (£)"
110 REM **Calculation loop**
120 FOR I = 1 TO T
130 PRINT I;"                ";D * (1 + P / 100) ^ I
140 NEXT I
150 REM **end of loop**
160 END
```

Program 20

```
RUN
Compound Interest Calculations

What is your deposit (£)?  500

What rate of interest (%)? 11.5

For how many years? 5

Time (years)    Yield (£)
1               557.5
2               621.612501
3               693.097938
4               772.804201
5               861.676685
Ready
```

[K] Program 20.

SAQ 2

Step no.	Line no.	N	S
1	20	0	0
2	30	1	0
3	50	1	1
4	70	2	1
5	50	2	5
6	70	3	5
7	50	3	14
8	70	4	14
9	50	4	30
10	70	5	30
11	50	5	55

SAQ 3

(a)

X	INT(X)	INT(X+0.5)
.4	0	0
.6	0	1
.8	0	1
1	1	1
1.2	1	1
1.4	1	1
1.6	1	2
1.8	1	2
2	2	2

(b)

X	INT(X)	INT(X+0.5)
0.2	0	0
0	0	0
−.2	0	0
−.4	−1	0
−0.6	−1	0
	−1	−1

SAQ 4

```
RUN

Using the ABS function

X =? 9
Y =? 14
A + Y = 23
ABS(X + Y) = 23

X =? 11
Y =? -2
X + Y = 9
ABS(X + Y) = 9

X =? -7
Y =? -8
X + Y = -15
ABS(X + Y) = 15

Ready
```

SAQ 5

9	N/G	$\frac{G + N/G}{2}$
1	12	6.5
6.5	1.8	4.15
4.15	2.89	3.52
3.52	3.41	3.46 etc.

Exercise 9

```
 10 REM **Cube root - iteration**
 15 CLS
 20 INPUT "No. for cube-rooting?";N
 30 PRINT
 40 INPUT "Accuracy required";A
 50 PRINT
 60 INPUT "Your guess ";G
 70 REM **start of iteration**
 80 REPEAT
 90 G = 0.5 * (G + (N / G ^ 2))
100 UNTIL ABS(N - G * G * G) < A
110 REM **Output**
120 PRINT
130 PRINT "The cube root of ";N
140 PRINT "is ";G
150 END
```

Program 21

```
RUN
No. for cube-rooting? 28

Accuracy required? 5E - 03

Your guess 3
The cube root of 28
is 3.03644268

Ready

RUN

No. for cube-rooting? 10101

Accuracy required? 5E - 03

Your guess 30

The cube root of 10101
is 21.6166388

Ready

RUN
No. for cube-rooting? -937

Accuracy required? 5E - 03

Your guess -10

The cube root of -937
is -9.78544136
```

🄺 Program 21.

UNIT 9
Introduction to data processing

9.1 Introduction

In this unit we shall emphasise again how important it is to impose some sort of order on data. In particular, we shall analyse in detail one method of ordering data: the interchange procedure for sorting. Having sorted the data we shall then show how to search through it quickly using the bisection search technique. Finally we shall look at how to handle data in tabular form.

These activities will also give us a chance to see how subroutines can help us perform many of those little repetitive tasks which can occur in programs of any size.

9.2 Sorting

In Unit 4 we spent some time discussing the procedure for finding the lowest value item in a list. We did this by an interchange procedure that put the lowest item into position 1 on the list. We said that we could repeat the procedure for the rest of the list, placing the second lowest value into position 2, etc., and we left you with the problem of sorting the whole list as an assignment. Because of the interchange sort's importance, we are now going to look at it in greater detail.

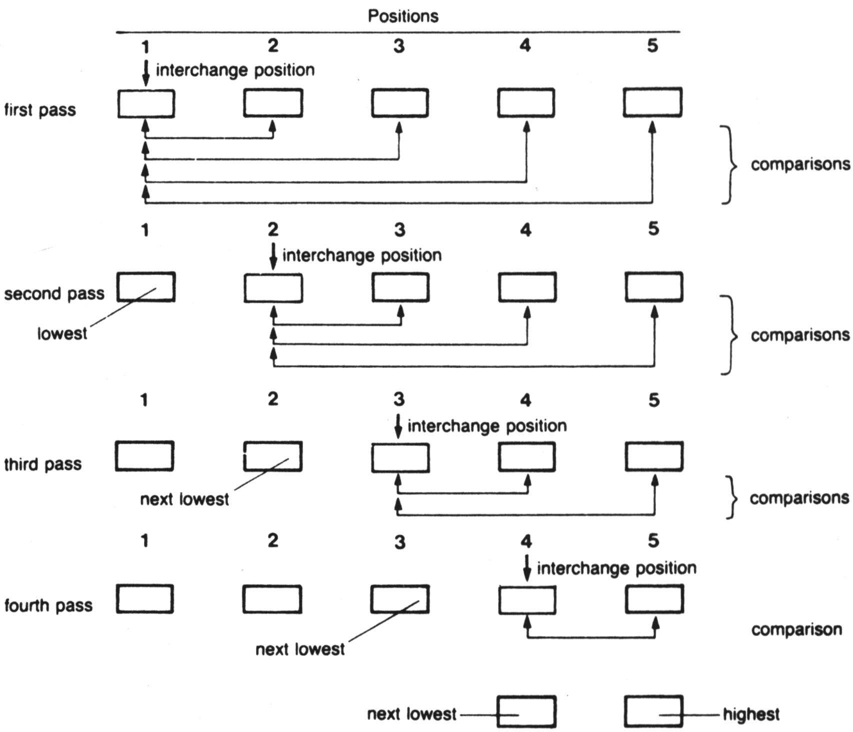


Figure 1 The sort procedure for a list of five items

Figure 1 illustrates the procedure for placing the items into locations 1 to 5 with the lowest item in 1, the next lowest in 2, and so on.

First pass. On the first pass all items are compared with the item in position 1 and the lowest is then placed in position 1.

Second pass. Position 1 can now be ignored and the procedure repeated on positions 2 to 5. This will find the next lowest which is placed in position 2.

Third pass. Now positions 1 and 2 can be ignored since they contain 'lowest' and 'next to lowest'. The procedure is repeated on positions 3 to 5. This finds the third lowest which goes in position 3.

Fourth pass. This is performed on items 4 and 5 only and results in the four lowest going into the fourth position. The remaining item must be the highest and will already be in the fifth position so no further passes are needed.

We can summarise the sort procedure as:

loop number	point interchange	remaining sub-sequence	
		start	end
1	1	2	5
2	2	3	5
3	3	4	5
4	4	5	5

Figure 2a Four sorts in a list of five items

Or, more generally, if we want to sort a list of N items:

loop number	interchange point	remaining sub-sequence	
		start	end
1	1	2	N
2	2	3	.
3	3	4	.
.	.	.	.
.	.	.	N
.	.	.	.
K	K	K+1	.
.	.	.	.
.	.	.	N
.	.	.	N
N−1	N−1	N	N

Figure 2b (N - 1) sorts in a list of N items

Since each pass involves a repetitive series of comparisons, it is an obvious candidate for a `FOR ... NEXT` loop. Then we need a further loop to decide which loop we are going round:

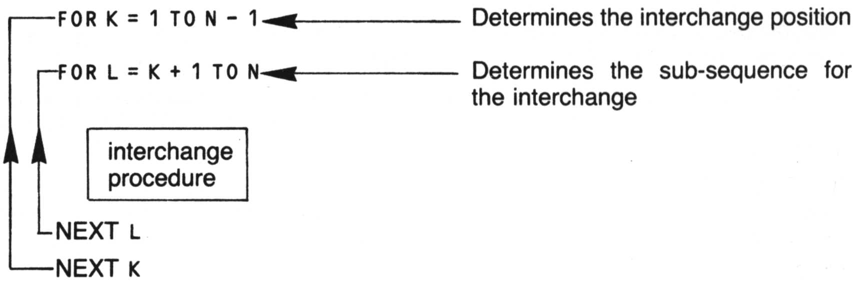


Figure 3 The nested loops of interchange sorting

SAQ 1

Use the interchange sort to place the following in order. Show the numbers stored at each location after each run.

6, 1, 4, 0, 2, 3, 7, 8

The program is:

```
210 REM **Sort Routine**
220 FOR K = 1 TO N - 1
230 FOR L = K + 1 TO N
240 IF X$(L) >= X$(K) THEN 275
250 T$ = X$(L)
260 X$(L) = X$(K)
270 X$(K) = T$
275 REM **Jump to here if items
276 REM    under scrutiny in right
277 REM    order**
280 NEXT L
290 NEXT K
300 REM **End of sort routine**
```

Outer loop decides interchange point

Inner loop decides the subsequence

If the item in the subsequence is >= the item in the interchange position, then do not interchange

The 'power house': should be condensed onto one line to speed up process

Program 1 Interchange sort

Using the sort program

The sort program can be used whenever it is needed. Here is one particular use: to sort a list of names into alphabetical order.

lines 50 to 100	read in the data
lines 210 to 300	carry out the sort
lines 410 to 460	print out the sorted list

```
 5 REM **Sorting Program**
 7 CLS
10 PRINT "..........Sort Routine.........."
12 PRINT
30 DIM X$(20)
50 LET I = 1
```

```
 55 REM **Data Input**
 60 INPUT "Next Data";X$(I)            Reading in the data
 70 IF X$(I) = "zzzz" THEN 185
 80 I = I + 1
100 GOTO 55
180 REM ****************************
185 REM **Length of list**
190 N = I - 1
200 REM ****************************
210 REM **Sort Routine**
220 FOR K = 1 TO N - 1
230 FOR L = K + 1 TO N
240 IF X$(L)> = X$(K) THEN 275
250 T$ = X$(L)
260 X$(L) = X$(K)                      Sort routine
270 X$(K) = T$
275 REM **Jump to here if items
276 REM   under scrutiny in right
277 REM   order**
280 NEXT L
290 NEXT K
300 REM **End of sort routine**
400 REM ****************************
410 PRINT
420 PRINT "Final Sorted List"
430 FOR P = 1 TO N
440 PRINT X$(P);" ";                   Printing out the result
450 NEXT P
460 PRINT
500 REM ****************************
```

Program 2 Using the sort routine

```
RUN
..........Sort routine..........

Next Data? Sam
Next Data? Bill
Next Data? Tony
Next Data? Pete
Next Data? Joe
Next Data? zzzz

Final Sorted List
Bill   Joe    Pete    Sam    Tony
Ready
```

9.3 Subroutines

By the time you have reached this stage you will begin to distinguish the wood from the trees. You will be aware that programs have an overall structure and are assemblies of smaller parts like the paragraphs of an essay. It is usual to break a program down into its constituent parts, and to write and test each part separately. Certain operations are often repeated several times throughout a program. The structure of a program may be simplified and tidied up by including these repetitive operations as subroutines.

We shall illustrate subroutines by taking a final look at the sort procedure. We are going to insert two extra trace print lines into the program so that we can see what is happening at each of the three stages of the sort routine:

Sort routine	**Trace print to show**
1. Input.	The list as taken in.
2. Sort.	The list after each sub-sequence.
3. Output.	The final, sorted, list.

Figure 4 shows the overall structure and how we can use one PRINT subroutine for all three PRINT operations.

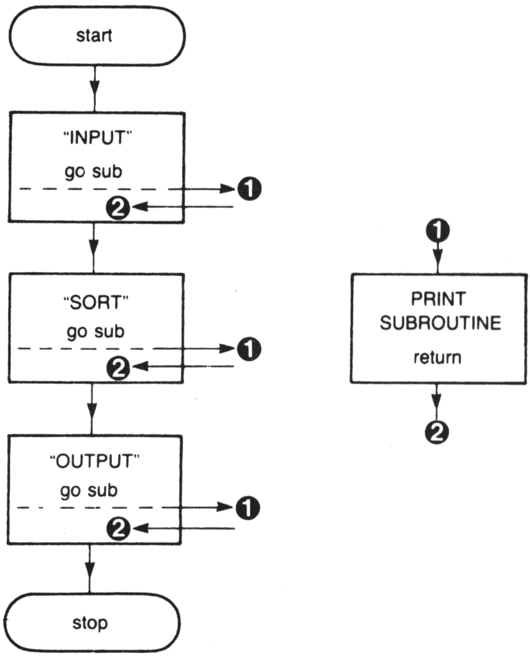


Figure 4 Print subroutine in the sort program

GO SUB

In BASIC to go into a subroutine we say:

```
GO SUB
```

followed by the line number of the start of the subroutine. Each subroutine must end with the statement

```
RETURN
```

which will return control to the next line in the main body of the program after the appropriate GO SUB statement. Thus in the following program segment line 30 transfers control to line 200, and lines 200 and 210 are executed. Then 220 returns

control to line 40 for the program to continue in its normal way.

```
  10 INPUT A
  20 INPUT B
  30 GOSUB 200
→ 40 LET S = A + B
  50 .....

 200 PRINT "A","B"  ←
 210 PRINT A,B            ]——————— GOSUB 200
 220 RETURN
```

SAQ 2

What is the value of B after this program has been run: (a) if 5 is inputted; (b) if 3 is inputted?

```
10 INPUT A
20 IF A < 5 THEN 40
30 GOSUB 70
35 REM **jump to here**
40 B = A * A
50 PRINT B
60 END
70 A = 1 / A
80 RETURN
```

Program 3

Here is the sort program with a print subroutine (lines 500-550) which is used each time the program executes line 195, line 280 and line 430.

```
  5 REM **Sorting Program**
  7 CLS
 10 PRINT "..........Sort Routine.........."
 12 PRINT
 30 DIM X$(20)
 50 I = 1
 55 REM **Data Input**
 60 INPUT "Next Data? ";X$(I)
 70 IF X$(I) = "zzzz" THEN 185
 80 I =I + 1
100 GOTO 55
180 REM ****************************
185 REM **Length of list**
190 N = I - 1
191 WAIT (100)
192 CLS
193 PRINT "List at start"
194 PRINT "*************"
195 GOSUB 510                →OUT  ]——— First trace
196 PRINT                    ←IN
200 REM ***************************
210 REM **Sort Routine**
220 FOR K = 1 TO N - 1
225 PRINT "Pass no. ";K
227 PRINT "**********"
230 FOR L = K + 1 TO N
240 IF X$(L) >= X$(K) THEN 275
250 T$ = X$(L)
260 X$(L) = X$(K)
270 X$(K) = T$
275 REM **Jump to here if items
276 REM   under scrutiny in right
277 REM   order**
280 GOSUB 510                →OUT  ]——— Second trace
285 NEXT L                   ←IN
287 PRINT
```

```
290 NEXT K
300 REM **End of sort routine**
400 REM *****************************
410 PRINT
420 PRINT "Final Sorted List"
425 PRINT "****************"
430 GOSUB 510                  →OUT  —— Output
450 END                        ←IN
500 REM **Print subroutine**  ┐
510 FOR P = 1 TO N            │
520 PRINT X$(P);" ";          ├—— Subroutine
530 NEXT P                    │
540 PRINT                     │
550 RETURN                    ┘
```

Program 4 Print subroutine in sort program

```
RUN
..............Sort Routine.............

Next Data? Tony
Next Data? Sam
Next Data? Pete
Next Data? Joe
Next Data? Bill
Next Data? zzzz
List at start
*************
Tony   Sam   Pete   Joe   Bill    ]—— Printed by GOSUB at line 195

Pass no. 1
**********
Sam     Tony    Pete    Joe     Bill

Pete    Tony    Sam     Joe     Bill

Joe     Tony    Sam     Pete    Bill

Bill    Tony    Sam     Pete    Joe

Pass no. 2
**********
Bill    Sam     Tony    Pete    Joe

Bill    Pete    Tony    Sam     Joe

Bill    Joe     Tony    Sam     Pete

Pass no. 3
**********
Bill    Joe     Sam     Tony    Pete

Bill    Joe     Pete    Tony    Sam

Pass no. 4
**********
Bill    Joe     Pete    Sam     Tony

Final Sorted List
****************
Bill    Joe     Pete    Sam     Tony]—— Printed by GOSUB at line 430

Ready
```

Each block printed by GOSUB at line 280 on the four occasions the program executes the loop controlled by K

Examples of subroutines

The purpose of a subroutine is to simplify and shorten long programs. By its very nature, then, it is difficult to get short meaningful programs which illustrate subroutines without their often being a little contrived. We need a program where the same or similar function is repeated at different points in the program.

Example 1

The game of dice ('craps' in the USA) provides a simple example. A pair of dice is thrown twice and the total score on each throw is noted. If the two scores are the same, the game ends. If they are different, the dice are thrown again. Write a program to simulate the game which prints out the number of throws required to obtain equal scores and what that score was.

Solution

```
  5 REM **Dice game simulation**
  7 CLS
 10 PRINT ".........2 Equal Throws........."
 20 PRINT
 30 C = 1
 35 REM **1st Throw**
 40 GOSUB 120
 50 S1 = s
 55 REM **2nd Throw**
 60 GOSUB 120
 70 S2 = S
 80 IF S1 = S2 THEN 100
 90 C = C + 1
 95 GOTO 35
100 REM **Printout**
105 PRINT "Equal Score ";S1;" in ";C;" throws"
110 END
120 REM **Dice rolling subroutine**
130 D1 = INT (RND(1) * 6) + 1)
135 D2 = INT (RND(1) * 6) + 1)
140 S = D1 + D2
150 RETURN
```

The subroutine produces generally different values of S for the 'main' program

Subroutine

Program 5 Simulation of 'craps'

```
RUN
.........2 Equal Throws.........
Equal Score 6 in 5 throws
Ready

RUN
.........2 Equal Throws.........
Equal score 10 in 4 throws
Ready

RUN
.........2 Equal Throws.........
Equal Score 11 in 1 throws
Ready

RUN
.........2 Equal Throws.........
Equal Score 8 in 6 throws
Ready

RUN
.........2 Equal Throws.........
Equal Score 9 in 17 throws
Ready
```

```
RUN
.........2 Equal Throws.........
Equal Score 8 in 2 throws
Ready

RUN
.........2 Equal Throws.........
Equal Score 8 in 10 throws
Ready
```

Exercise 1

(a) Write a segment of program to print a line of dashes "--------" across the screen or printer, 38 spaces for the screen.

(b) Write one line of program to print a 'submarine' or <=> at any point across the screen or printer where the variable S determines the position.

(c) Write a program to print on successive lines:
   (i) a line of dashes;
   (ii) a submarine at any point;
   (iii) another line of dashes;
   with the line printing in (i) and (iii) in a subroutine.

Exercise 2

Now you must admit that the solution to Exercise 1 looks like a vessel in a canal, so why not a submarine as we are concerned with subroutines? Instead of battleships in a two-dimensional sea, we have a submarine in a one-dimensional canal. Anyway, we have the picture for a simple game.

Write a program to generate a random number between 1 and 33 for a screen. The submarine is going to take up the last three positions of the width. Use the random number to print the submarine in random positions along the canal.

Exercise 3

The essence of the game we are going to play with the machine will be clear from Exercise 2. The computer generates a random number and invites you to find the submarine by guessing a number between 1 and 33. If you guess the correct position, i.e. between S and S + 2 if S is the random number (remember the submarine takes up three places in the line), then the machine records a 'hit' and the games ends. If you don't find the submarine, the machine will record a 'miss' and invite you to try again. Write a program to do this.

(We advise you to write the program to give yourself the option to stop playing before you find the submarine, because it is infuriating to have to try every position across the screen just to stop the program running. You could just pull the plug out, and then it would sink!?)

9.4 Searching

The submarine problem gives us a good lead into discussions about searching data. The only methodical way to find the submarine was to search the canal successively position by position starting from one end. How much easier it would have been had the program responded with 'too high' or 'too low', as appropriate, after each guess. No doubt you can immediately think of a procedure for 'homing-in' on the submarine as quickly as possible!

Similarly, if dictionaries, telephone directories, encyclopedias and library catalogues were not arranged in alphabetical order, think how difficult it would be to find the desired information.

But if we've gone to a lot of trouble to sort our data into numerical or alphabetical order, then we need an efficient search technique to find any given item. If we consult a dictionary or telephone directory for an item, we don't start looking at the first page and work methodically through the volume page by page until the item is found. We take a rough guess, e.g. if the name begins with 'P' then we try to open the directory at just over half-way through it, and start looking from that point.

Bisection search

A 'rough' guess is too imprecise a term for a computer. However we can specify guessing points as follows:

- divide the range of items into half and ask 'is the item above or below the half-way mark?'
- if it is below then we define a new range with the middle item now acting as the upper limit;
- if above then the middle item becomes our lower limit;
- either way we discard half the old range and repeat our halving or bisection procedure with the new range.

So the bisection search is, in outline:

Is 7 in the list 1,2,3,4,5,6,7,8,9,10?

List in order: [1 2 3 4 5 6 7 8 9 10]

Halve list. [1 2 3 4 5 | 6 7 8 9 10]

Is 7 = middle item? No.
Is 7 < middle item? No.
So 7 is in top half.

Halve list. [6 7 8 9 10]

Is 7 = middle item? No.
Is 7 < middle item? Yes.

Halve list. [6 7 8]

Is 7 = middle item? Yes.
So 7 is in list.

That outline illustrates the principle of the bisection search but in practice we need to distinguish between the values of the items in a list and the indexes of those items.

Example 2

An ordered list contains the items A, F, I, M, P, T, U, Z. Use the bisection search procedure to find whether or not P is in the list.

We call P, Query – the value we wish to enquire about.

Index	1	2	3	4	5	6	7	8
Item	A	F	I	M	P	T	U	Z
Start-Index	Low (1)							High (8)
Mid-Index, $\text{Int}\left(\frac{1+8}{2}\right)=4$				Mid (4)				

Comparisons

> is Query = Item (4)? no!
> is Query < Item (4)? no!
> make Index (4) the new Low

Index	4	5	6	7	8
Item	M	P	T	U	Z
Start-Index	Low (4)				High (8)
Mid-Index, $\text{Int}\left(\frac{4+8}{2}\right) = 6$			Mid(6)		

Comparisons

> is Query = Item (6)? no!
> is Query < Item (6)? yes!
> make Index(6) the new High

Index	4	5	6
Item	M	P	T
Start-Index	Low(4)		High(6)
Mid-Index, $\text{Int}\left(\frac{4+6}{2}\right) = 5$		Mid(5)	

Comparison

> is Query = Index (5)? yes!

Figure 5 Bisection search

Exercise 4
Carry out the bisection search procedure on the list in Example 2 but looking for the letter I.

We only had to make three comparisons to home in on the item 'P' in Example 2, but they were a bit long winded, and the whole procedure may seem to have little advantage over simply searching straight through the list. The effectiveness of the method is not really apparent in short lists. We will demonstrate its power in seaching longer lists later, but first we still have some loose ends to tie up.

Some problems with bisection search
(a) How to stop

Example 3
Carry out the same procedure as before, but search for the letter 'Q'.

The method would proceed exactly the same as before as far as the third comparison, so we'll pick up the story there.

Query = Q

Index	4	5	6
Item	M	P	T
Start-Index	Low(4)		High(6)
Mid-Index, $\text{Int}\frac{(4 + 6)}{2} = 5$		Mid (5)	

Comparisons

is Query = Item(5)? no! is Query < Item(5)? no! make Index(5) the new Low

Index	4	5	6
Item		P	T
Start-Index		Low(5)	High(6)
Mid-Index, $\text{Int}\frac{(5 + 6)}{2} = 5$		Mid (5)	

Comparisons

is Query = Item(5)? no! is Query < .....we have done this before?!

So we don't seem able to stop. Q is not there but we are stuck looking for it between P and T. We have already met the problem of stopping the process in the last example. If the indexes Low and High have moved so close that they are at adjacent positions, and Query is not yet found, then Query is not a member of the list. That is the end and outcome of the search. So the end is either when the Query has been found, or when Low and High occupy adjacent indexes (High - Low = 1).

(b) How to start

To start the process seemed straightforward enough. We make Index(1) = Low and Index(N) = High. Trouble would occur however if Item(1) and Item(N) were not the lowest and highest possible values.
E.g. consider the following list which does not include letters before C or after S:

1	2	3	4	5
C	F	G	P	S
Low				High

If Query was A or B or higher than S, then the process would not work. The easiest solution is to ensure that the items at the ends of the list will always have the extreme values, e.g. in a list of names make Item(1) = aaaaaa and Item(N) = zzzz.

We will now outline the algorithm in flowchart form.

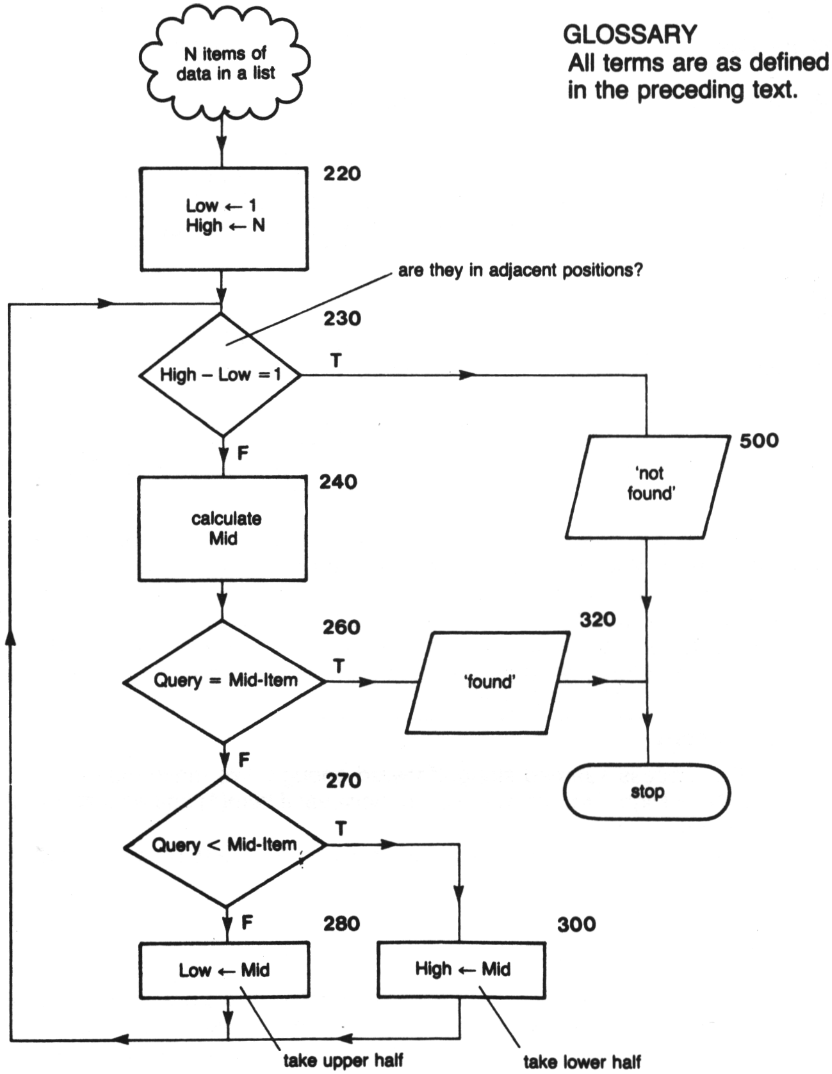


Figure 6 Flowchart for bisection search

All we have to do now is to write the program:

```
10 REM **Bisection Search**
15 CLS
20 DIM N$(20)
30 DIM T$(20)
40 I = 1
45 REM **Read in name & no.**
50 READ N$(I)
55 READ T$(I)
60 IF N$(I) = "zzzz" THEN 90
70 I = I + 1
80 GOTO 45
90 REM ****************************
100 N = I
105 REM **We found "zzzzz" this time**
110 REM ****************************
140 PRINT
150 INPUT "Query name? ";Q$
200 REM **start of search**
210 PRINT " L";TAB (5);" H";TAB (10);" M";
    TAB (15);"N$(M)"
220 L = 1
225 H = N
230 REM ****************************
235 IF H - l = 1 THEN 500
240 M = INT((L + H) / 2)
250 PRINT L;TAB (5);H;TAB (10);M;TAB (15);N$(M)
260 IF Q$ = N$(M) THEN 320
270 IF Q$ < N$(M) THEN 300
280 L = M
290 GOTO 230
300 H = M
310 GOTO 230
320 REM **End of search**
330 PRINT Q$;"'s tele no. is ";T$(M)
350 GOTO 600
500 REM **Name not found**
510 PRINT Q$;" is not in the list"
600 REM **Continue or not?**
605 INPUT "Do you wish to look for another name? ";R$
610 IF R$ = "yes" THEN 110
890 DATA aaaaaa,0000
900 DATA Barrow,1234
910 DATA Copper,9832
920 DATA Draper,1980
930 DATA Edward,7294
950 DATA Gwynne,5821
960 DATA Hettie,8632
970 DATA Morley,7832
980 DATA Peters,1383
990 DATA Smythe,1147
1000 DATA Weekes,5529
1010 DATA Wilson,9936
1020 DATA zzzz..9999
1050 END
```

Program 6 Bisection Search (with TAB)

210 **and 250** are trace printlines only

```
10 REM **Bisection Search**
15 CLS
20 DIM N$(20)
30 DIM T$(20)
40 I = 1
45 REM **Read in name and no.**
50 READ N$(I)
55 READ T$(I)
60 IF N$(I) = "zzzz" THEN 90
70 I = I + 1
80 GOTO 45
90 REM ****************************
100 N = I
105 REM **We found "zzzz" this time**
110 REM ****************************
140 PRINT
150 INPUT "Query name";Q$
200 REM **start of search**
210 PRINT "L    H     M    N$(M)"
220 L = 1
225 H = N
230 REM ****************************
235 IF H - L = 1 THEN 500
240 M = INT((L + H) / 2)
250 PRINT L;
252 IF L < 10 THEN PRINT " ";
254 PRINT "   ";H;
256 IF H < 10 THEN PRINT " ";
258 PRINT "   ";M;
260 IF M < 10 THEN PRINT " ";
262 PRINT "   ";N$(M)
265 IF Q$ = N$(M) THEN 320
270 IF Q$ < N$(M) THEN 300
280 L = M
290 GOTO 230
300 H = M
310 GOTO 230
320 REM **End of search**
330 PRINT Q$;"'s tel no. is ";T$(M)
350 GOTO 600
500 REM **Name not found**
510 PRINT Q$;" is not in the list"
600 REM **Continue or not?**
605 INPUT "Do you wish to look for another name";R$
610 IF R$ = "yes" THEN 110
890 DATA aaaaaa,000
900 DATA Barrow,1234
910 DATA Copper,9832
920 DATA Draper,1980
930 DATA Edward,7294
950 DATA Gwynne,5821
960 DATA Hettie,8632
970 DATA Morley,7832
980 DATA Peters,1383
990 DATA Smythe,1147
1000 DATA Weekes,5529
1010 DATA Wilson,9936
1020 DATA zzzz,9999
1050 END
```

Program 6 Bisection search (without TAB)

```
RUN
Query name? Morley
L    H     M     N$(M)
1    13    7     Hettie
7    13    10    Smythe
7    10    8     Morley
Morley's tele no. is 7832

Do you wish to look for another name? yes

Query name? Wilson
L    H     M     N$(M)
1    13    7     Hettie
7    13    10    Smythe
10   13    11    Weekes
11   13    12    Wilson
Wilson's tele no. is 9936

Do you wish to look for another name? yes

Query name? Weeks
L    H     M     N$(M)
1    13    7     Hettie
7    13    10    Smythe
10   13    11    Weekes
11   13    12    Wilson
Weeks is not in the list

Do you wish to look for another name? no
Ready
```

9.5 Tables

When we want to store a lot of information there are various methods open to us. One is using lists (see Unit 4), which are sometimes called one-dimensional arrays. A second method is to use tables or two-dimensional arrays.

Suppose you want to store the following data:

	1st qtr	2nd qtr	3rd qtr	4th qtr
Car sales	20	70	80	40
Servicing	10	14	18	11
Petrol	30	45	50	30

Figure 7 Income for Main Road Service Station (£,000's)

Now you could put this in one list but it would be hard to use. The first four items would be income for car sales, the next four for servicing, etc. Alternatively you would have three lists: one for car sales, one for servicing and one for petrol. But BASIC allows you to have a two dimensional table named by any of the 286 variable names, e.g.

T(,)

Comparison of lists and tables

Lists need one index to describe a position in the list. Tables need two, which are usually called sub-scripts, not indices (or indexes, as we have called them).

List

L(1),L(2),L(3) ... L(I) ...

index of this item = 3

Array

A(1,1) A(1,2) A(1,3)
A(2,1) A(2,2) A(2,3)
A(3,1) A(3,2) A(3,3)

this item needs two sub-scripts:
3 to tell us it is in row 3;
2 to tell us it is in column 2.

Tables

- A table must contain either all string variables or all numerical variables. (Numbers can of course be stored as strings, and their values found by the VAL-function.)
- We use one of the 286 variable names allowed in our minimal-BASIC scheme to describe the table as a whole, e.g. a table, B$ table, M$ table.

Generally, a table comprises:

	col.1	col.2	col.3	col.4
row 1	r1c1	r1c2	r1c3	....
row 2	r2c1	r2c2	r2c3	....
row 3	r3c1	r3c2	r3c3	....
etc	....	....	....	....

Figure 8 The rows and columns of a table

For the service station data, T needs three rows and four columns and so contains 12 items:

```
T(1,1)   T(1,2)   T(1,3)   T(1,4)
T(2,1)   T(2,2)   T(2,3)   T(2,4)
T(3,1)   T(3,2)   T(3,3)   T(3,4)
```

So

T(2,1) = 10

T(3,3) = 50 etc.

This is very similar to the idea of tables which you met previously. There we said that a file consists of a series of records each of which consists of fields. In table form this would look like:

	Field 1	Field 2
	Name	Telephone number
Record 1	Benny	1234
Record 2	Copper	9823
Record 3	Draper	1850
Record 4	Eddie	7294

Or, more generally:

	Field 1	Field 2	Field 3	etc.
Record 1	R1F1	R1F2	R1F3	....
Record 2	R2F1	R2F2	R2F3	....
Record 3	R3F1	R3F2	R3F3	....
etc.	....	....	....	....

If the telephone numbers table is called T$ then the individual items will be labelled:

	Field 1	Field 2
	Name	Telephone number
Record 1	T$(1,1) = Benny	T$(1,2) = 1234
Record 2	T$(2,1) = Copper	T$(2,2) = 9823
Record 3	T$(3,1) = Draper	T$(3,2) = 1950
Record 4	T$(4,1) = Eddie	T$(4,2) = 7294

- The whole table is called T$ table.
- Each item in the table is described by two subscripts. Thus 1950 (3rd row, 2nd column) is

  T$(3,2)

• The 3 and 2 describe the position of item T$(3,2), not its value. Its value is 1950.

So we say

T$(3,2) = 1950

> In general
>
> T$(R,C)
>
> Row subscript — Column subscript

Example 4

This N$-table has nine values as shown. What are their variable names?

R \ C		1	2	3
N$(,)	1	Benny	Copper	Draper
	2	Eddie	Gwynne	Hetty
	3	Morley	Prosser	Smythe

Solution

Benny = N$(1,1) Copper = N$(1,2) Draper = N$(1,3)

Eddie = N$(2,1) Gwynne = N$(2,2) Hetty = N$(2,3)

Morley = N$(3,1) Prosser = N$(3,2) Smythe = N$(3,3)

SAQ 3

In the following A$ table identify the variables and their values as in Example 4.

Archer	Benny	Copper
Draper	Eddie	Frame
Gwynne	Hetty	Kemp
Lamb	Morley	Noakes
Prosser	Smythe	Tait

Tables and nested loops

If `FOR ... NEXT` loops and lists seemed to be made for each other, then even more so do nested `FOR ... NEXT` loops and tables seem complementary.

For example, suppose you want to place:

Archer,Benny,Copper,Draper,Eddie,Frame,Gwynne,Hetty,Kemp,Lamb, Morley,Noakes, Prosser,Smythe,Tait,Weekes

into a table, N$, with four rows and four columns. (We need a string array because we are storing string data.) This can be done with an `INPUT` statement in two nested loops:

```
 60 FOR I = 1 TO 4
 70 FOR J = 1 TO 4
 80 INPUT "Next input";N$(I,J)
 83 PRINT TAB (30);4 * (I - 1) + J
 90 NEXT J
100 NEXT I
```

This process is carried out in full by lines 10 to 100 of Program 7.

It's all very well to store the values in a table, but of course we cannot see the result of this until we print it out. The second half of the program prints the table values out in a column with I and J accompanying them so that you can identify clearly how I and J are used.

```
  5 REM **4 x 4 Array**
  7 CLS
 10 PRINT ".....Table Input and Print....."
 15 PRINT
 20 PRINT "Table set up for 16 inputs..."
 30 DIM N$(20,5,)———————————————DIM statement for 2-D array.
 40 REM **Input Routine**        We are asking for space for
 60 FOR I = 1 TO 4               20 rows and 5 columns.
 70 FOR J = 1 TO 4
 80 INPUT "Next input?";N$(I,J)
 82 PRINT N$(I,J);
 83 PRINT TAB(30);4 * (I - 1) + J
 90 NEXT J
100 NEXT I
110 REM **Print Routine**
112 PRINT ".....Table Full....."
113 WAIT(100)
115 CLS
116 PRINT "I";TAB(10);"J";TAB(20);"N$(I,J)"
130 FOR I = 1 TO 4 ——————————————————————— rows
140 FOR J = 1 TO 4 ——————————————————————— columns
150 PRINT I;TAB(10);J;TAB(20);N$(I,J)
160 NEXT J
180 NEXT I
200 END
```

Program 7 Data in 4×4 Array (with TAB)

```
  5 REM ** 4 x 4 Array**
  7 CLS
 10 PRINT ".........Table Input and Print........."
 15 PRINT
 20 PRINT "Table set up for 16 inputs..."
 30 DIM N$(20,5)
 40 REM ** Input Routine**
 60 FOR I = 1 TO 4
 70 FOR J = 1 TO 4
 80 INPUT "Next input";N$(I,J)
 82 PRINT N$(I,J)
 83 PRINT "
 90 NEXT J
100 NEXT I
110 REM **Print Routine**
112 PRINT ".........Table Full........."
113 WAIT(100)
115 CLS
116 PRINT"I            J           N$(I,J)"
130 FOR I = 1 TO 4
140 FOR J = 1 TO 4
150 PRINT I;"          ";J;"           ";N$(I,J)
160 NEXT J
180 NEXT I
200 END
```

Program 7 (without TAB)

```
RUN
(input section omitted)

I              J              N$(I,J)
1              1              Archer
1              2              Benny
1              3              Copper
1              4              Draper
2              1              Eddie
2              2              Frame
2              3              Gwynne
2              4              Hetty
3              1              Kemp
3              2              Lamb
3              3              Morley
3              4              Noakes
4              1              Prosser
4              2              Smythe
4              3              Tait
4              4              Weekes
Ready
```

SAQ 4

The following amendments are made to Program 7. Write out what the output table will look like.

```
 60 FOR I = 1 TO 3
 70 FOR J = 1 TO 5
130 FOR I = 1 TO 3
140 FOR J = 1 TO 5
```

Table output

The output of Program 7 is not very satisfactory since we want to see the table in table form. To do this we delete line 116 and insert a new print routine:

```
  5 REM **5 x 3 Array**
  7 CLS
 10 PRINT ".....Table Input and Print....."
 15 PRINT
 20 PRINT "Table set up for 15 inputs..."
 30 DIM N$(20,5)
 40 REM **Input routine**
 60 FOR I = 1 TO 5
 70 FOR J = 1 TO 3
 80 INPUT "Next input";N$(I,J)
 82 PRINT N$(I,J);
 83 PRINT "                        ";4 * (I - 1) + J
 90 NEXT J
100 NEXT I
110 REM **Print Routine**
112 PRINT ".....Table Full....."
113 WAIT(100)
115 CLS
130 FOR I = 1 TO 5
140 FOR J = 1 TO 3
150 PRINT TAB (10 * (J - 1));N$(I,J);
160 NEXT J
170 PRINT
180 NEXT I
200 END
```

The first column starts at position 1 - 1 = 0

Columns ten characters wide

Program 8 (with TAB)

```
  5 REM ** 5 x 3 Array**
  7 CLS
 10 PRINT ".........Table Input and Print........."
 15 PRINT
 20 PRINT "Table set up for 15 inputs..."
 30 DIM N$(20,5)
 40 REM **Input Routine**
 60 FOR I = 1 TO 5
 70 FOR J = 1 TO 3
 80 INPUT "Next input";N$(I,J)
 82 PRINT N$(I,J);
 83 PRINT "                         ";4 * (I - 1) + J
 90 NEXT J
100 NEXT I
110 REM **Print Routine**
112 PRINT ".........Table Full........."
113 WAIT(100)
115 CLS
130 FOR I = 1 TO 5
140 FOR J = 1 TO 3
150 PRINT N$(I,J)
155 FOR X = 1 TO 10 - LEN(N$(I,J))
157 PRINT " ";
159 NEXT X
160 NEXT J
170 PRINT
180 NEXT I
200 END
```

Program 8 (without TAB)

The output then is:

```
Archer      Benny       Copper
Draper      Eddie       Frame
Gwynne      Hetty       Kemp
Lamb        Morley      Noakes
Prosser     Smythe      Tait
```

9.6 Using the printer

There are two commands in Oric BASIC to control a printer: `LPRINT` and `LLIST`.

`LPRINT` acts in the same way as `PRINT`, except that it prints onto the printer paper, instead of on the display screen, so

`LPRINT` on its own prints a blank line

`LPRINT` with quotation marks (" ") can be used to output messages on the printer paper either as a direct command, or as an integral part of a program.

`LLIST` enables you to print out a listing of your program currently in the computer's memory.

Some BASICs support a statement `LRUN`, which allows you to copy the screen display onto the printer, but Oric BASIC does not recognise this command.

9.7 Multiple statement program lines

The Oric has the capacity to allow more than one statement on each numbered program line:

```
10 FOR X = 1 TO 10: PRINT "Line Number ";X: NEXT X
```

takes up less space on a page than

```
10 FOR X = 1 TO 10
20 PRINT "Line Number ";X
30 NEXT X
```

But it is no more efficient, in programming terms, and can make a program listing much more difficult to follow. Thus, we have avoided using multiple statement lines in this text.

There is one case in which multiple statements may be helpful; this is when you put a `REM` statement as the second statement on the line, and that `REM` statement refers directly to the first statement on the line, for example

```
100 IF Q$= "5" THEN 50:REM reject touching a number more than 5
```

or

```
210 LET N = N - 1: REM don't count zzzz as an input record
```

9.8 Use of 'menus'

The following program segment is a routine from a much larger program, which provides explanations and tests on various mathematical ideas.

Lines 310 to 380 produce the display which gives you a choice of four options: this is the 'menu' from which you choose. This is the display:

```
          CHOOSE A NUMBER
          ---------------

          1.  What is VOLUME?

          2.  VOLUME sums.

          3.  CAPACITY sums.

          4.  Stop the program.

   Touch the number then RETURN key.
```

Note that we have used the `PLOT` statement to lay out the display in as few lines as possible. (See Unit 5). Lines 372 to 376 move the cursor down to line 18 (where the `INPUT` statement is), as `PLOT` statements do not move the cursor position. Lines 390 to 450 check that the input only contains numbers. If the value 0 is input, the menu display is cycled, just to let the user know that he went wrong. Lines 490 to 550 check to see if a number greater than 4 was entered, and if this is so, a little message is displayed for a fixed time, then the screen is cleared and the menu redisplayed. These validation procedures make the program crashproof, whilst remaining user-friendly.

This is the display when a number larger than 4 is entered:

```
            CHOOSE A NUMBER:
            ----------------

            1.  What is VOLUME?

            2.  VOLUME sums.

            3.  CAPACITY sums.

            4.  Stop the program.

    Only numbers 1 to 4, please!  5
```

The real crux of the whole routine is at lines 560 and 600; now that we have arrived at a choice of 1, 2, 3 or 4, we can use another feature of Oric BASIC, the `ON ... GOTO ...` statement. We have stored the valid menu choice in a variable `Q`, and line 600 reads:

```
600 On Q GOTO 1000,2000,3000,4000
```

which means that if `Q` is 1, program control goes to line 1000, if `Q` is 2 to 2000, if `Q` is 3 to 3000, and if `Q` is 4 to 4000. There is no danger of confusion because of any other value in `Q`, as we have taken care to validate the contents of `Q`.

Some BASICs allow a computed `ON ... GOTO ...`, e.g.

```
600 ON Q GOTO Q * 1000
```

which is more concise than our line 600 – however, this is not a valid option in Oric BASIC.

Obviously, if the whole program were listed, between 1000 and 1990 would be the explanation of volume as a maths topic, with relevant examples, not just a simple message! Line 1990 then returns you to the menu, to allow a further choice to be made. Note the need for a stop option – all other choices automatically return you to the menu routine after operation.

```
300 REM **menu routine**
310 CLS
320 PLOT 11,4,"Choose a number:"
330 PLOT 10,5,"----------------"
340 PLOT 10,8,"1.  What is VOLUME?"
350 PLOT 10,10,"2.  VOLUME sums."
360 PLOT 10,12,"3.  CAPACITY SUMS."
370 PLOT 10,14,"4.  Stop the program."
372 FOR X = 1 TO 18
374 PRINT
376 NEXT X
380 INPUT "Touch the number then RETURN key.";Q$
390 FOR I = 1 TO LEN(Q$)
400 IF MID$(Q$,I,1) < "0" OR MID$(Q$,I,1) > "9" THEN 310
410 NEXT I
420 LET Q = VAL(Q$)
430 IF Q > 0 THEN 480
440 REM **Cycle Menu**
450 CLS
460 WAIT(100)
470 GOTO 320
480 REM **check menu**
490 IF Q < 5 THEN 550
```

```
 500 REM **invalid number**
 510 PLOT 1,18,"   Only numbers 1 to 4, please! "
 520 WAIT(200)
 530 CLS
 540 GOTO 320
 550 REM **valid choices**
 560 CLS
 600 ON Q GOTO 1000,2000,3000,4000
1000 REM **volume definition**
1010 CLS
1020 PRINT "You would put information here."
1030 WAIT(300)
1990 GOTO 300
2000 REM **volume sums**
2010 CLS
2020 PRINT "You would put sums routine here."
2030 WAIT(300)
2990 GOTO 300
3000 REM **capacity definition**
3010 CLS
3020 PRINT "You would put sums routine here."
3030 WAIT(300)
3990 GOTO 300
4000 REM **end routine**
4010 CLS
4020 PRINT
4030 PRINT "Thank you - bye now!"
4990 END
```

Program 9

[K] Program 9.

Assignment 9

1. A salesman has four product lines. The value (in £) of his firm orders for one week are shown in the table.

day \ product	1	2	3	4	totals
1	500	300	20	25	e
2	600	700	40	0	f
3	200	550	60	20	g
4	250	450	100	5	h
5	400	200	100	11	i
totals	a	b	c	d	t

   Write a program which will help him analyse his week's work by giving:

   (i) his day totals (e,f,g,h,i);
   (ii) his product totals (a,b,c,d);
   (iii) his overall weekly total (t).

2. Write a program to extend the submarine game to a 10 × 10 grid. If the guess is close to the submarine then the program should give a 'near miss' clue. You decide what is meant by 'close'.

Objectives of Unit 9

Now that you have completed this unit, check that you are able to:

Use the interchange sort (manually) on a set of data.

Write two nested program loops to perform the interchange sort. □

Follow `GOSUB` in programs. □

Write `GOSUB` into programs. □

Use the bisection search (manually) on a set of data. □

Write a program for the bisection search. □

Put data into two dimensional arrays. □

Write a program to read data into a two dimensional array. □

Write a program to print data out of a two dimensional array. □

Write a program to find the row sums and the column sums in a two dimensional array. □

Use of 'menu' routines (with `ON ... GOTO ...`) □

Answers to SAQs and Exercises

SAQ 1

```
After 1st run        0  1  4  6  2  3  7  8
After 2nd run        0  1  4  6  2  3  7  8
After 3rd run        0  1  2  6  4  3  7  8
After 4th run        0  1  2  3  4  6  7  8
```

SAQ 2

(a) B = 1/25 Shown as .04

(b) B = 9 (`GOSUB` is not used in this case.)

Note: It's best to get rid of the cursor with `CTRL-Q` before Exercises 1–3.

Exercise 1

(a)

```
 5 REM **Exercise 1**
 7 CLS
10 FOR I = 1 TO 38
20 PRINT "-";
30 NEXT I
40 PRINT
50 END
```

Program 10

(b)

```
PRINT TAB(S);"<=>"
```

(c)

```
 5 REM **Exercise 1**
 7 CLS
10 INPUT S
20 GOSUB 100
30 PRINT TAB(S);"<=>"
40 GOSUB 100
50 END
```

```
100 REM **draw 'canal'**
110 FOR I = 1 TO 38
120 PRINT "-";
130 NEXT I
140 PRINT
150 RETURN
```

Program 11 (with TAB)

```
  5 REM **Exercise 1**
  7 CLS
 10 INPUT S
 20 GOSUB 100
 30 FOR X = 1 TO S
 32 PRINT " ";
 34 NEXT X
 36 PRINT "<=>"
 40 GOSUB 100
 50 END
100 REM **draw 'canal'**
110 FOR I = 1 TO 38
120 PRINT "-";
130 NEXT I
140 PRINT
150 RETURN
```

Program 11 (without TAB)

The value that we give S will determine the position of the submarine along the canal, and we get a picture like:

```
--------------------------------
                        <=>
--------------------------------
```

Exercise 2

```
  5 REM **Exercise 2**
  7 CLS
 10 S = INT(RND(1) * 33) + 1
 20 GOSUB 100
 30 PRINT TAB(S);"<=>"
 40 GOSUB 100
 50 END
100 REM **draw 'canal'**
110 FOR I = 1 TO 38
120 PRINT "-";
130 NEXT I
140 PRINT
150 RETURN
```

Program 12

```
  5 REM **Exercise 2**
  7 CLS
 10 S = INT(RND(1) * 33) + 1
 20 GOSUB 100
 30 FOR X = 1 TO S
 32 PRINT " ";
 34 NEXT X
 36 PRINT "<=>"
 40 GOSUB 100
 50 END
100 REM **draw 'canal'**
110 FOR I = 1 TO 38
120 PRINT "-";
130 NEXT I
140 PRINT
150 RETURN
```

Program 12 (without TAB!)

```
RUN

--------------------------------
             <=>
--------------------------------

Ready
RUN

--------------------------------
    <=>
--------------------------------

Ready
RUN

--------------------------------
                       <=>
--------------------------------

Ready
```

Exercise 3

```
 10 REM **Submarine**
 20 CLS
 30 REM **Print challenge**
 40 GOSUB 300
 50 PRINT "A number from 1 to 33 might find me"
 60 GOSUB 300
 70 REM **Random position of sub**
 80 S = INT(RND(1) * 33 + 1)
 85 REM **next go**
 90 PRINT
100 INPUT "Try another number";X
110 IF X < S THEN 190
120 IF X > S + 2 THEN 190
130 REM **a hit**
140 GOSUB 300
150 PRINT TAB(S);"Hit"
160 GOSUB 300
170 GOTO 320
180 REM **a miss**
190 GOSUB 300
200 PRINT "You missed"
210 GOSUB 300
220 INPUT "Do you still want to play ";R$
230 IF R$ = "Yes" THEN 85
235 PRINT
240 PRINT "Spoilsport!! I was here"
245 PRINT
250 GOSUB 300
260 PRINT TAB(S);"<=>"
270 GOSUB 300
280 GOTO 320
290 REM **Print subroutine**
300 FOR I = 1 TO 38
305 PRINT "-";
310 NEXT I
311 PRINT
315 RETURN
320 END
```

Program 13 (with TAB)

```
 10 REM **Submarine**
 20 CLS
 30 REM **Print challenge**
 40 GOSUB 300
 50 PRINT "A number from 1 to 33 might find me"
 60 GOSUB 300
 70 REM **Random position of sub**
 80 S = INT(RND(1) * 33) + 1
 85 REM **next go**
 90 PRINT
100 INPUT "Try another number";X
110 IF X < S THEN 190
120 IF X > S + 2 THEN 190
130 REM **a hit**
140 GOSUB 300
150 GOSUB 350
155 PRINT "Hit"
160 GOSUB 300
170 GOTO 320
180 REM **a miss**
190 GOSUB 300
200 PRINT "You missed"
210 GOSUB 300
220 INPUT "Do you still want to play";R$
230 IF R$ = "Yes" THEN 85
235 PRINT
240 PRINT "Spoilsport!! I was here"
245 PRINT
250 GOSUB 300
260 GOSUB 350
265 PRINT "<=>"
270 GOSUB 300
280 GOTO 320
290 REM **Print subroutine**
300 FOR I = 1 TO 38
305 PRINT "-";
310 NEXT I
311 PRINT
315 RETURN
320 END
350 REM **spacing**
360 FOR X = 1 TO S
370 PRINT " ";
380 NEXT X
390 RETURN
```

Program 13 (without TAB)

```
RUN
--------------------------------------
A number from 1 to 33 might find me
--------------------------------------

Try another number? 4

--------------------------------------
You missed
--------------------------------------

Do you still want to play? Yes
Try another number? 7

--------------------------------------
You missed
--------------------------------------
```

```
Do you still want to play? Yes
Try another number? 19

------------------------------------
You missed
------------------------------------

Do you still want to play? n
Spoilsport!! I was here
---------------------------------
                        <=>
---------------------------------
Ready
```

Exercise 4

1	2	3	4	5	6	7	8
A	F	I	M	P	T	U	Z

•
•
•
Mid(4)

Query = Item(4)? No.
Query < Item(4)? Yes.
make Index(4) the new high

A	F	I	M
1	2	3	4

$$\text{Mid-Index} = \text{Int}\frac{(1 + 4)}{2} = 2$$

•
•
•

Query = Item(2)? No.
Query < Item(2)? No.
make Index(2) the new low

2	3	4
F	I	M

•
•
•

Query = Item(3)? Yes.
Therefore I is in list

SAQ 3

A$(1,1) = Archer
A$(2,1) = Draper
A$(3,1) = Gwynne
A$(4,1) = Lamb
A$(5,1) = Prosser

A$(1,2) = Benny
A$(2,2) = Eddie
A$(3,2) = Hetty
A$(4,2) = Morley
A$(5,2) = Smythe

A$(1,3) = Copper
A$(2,3) = Frame
A$(3,3) = Kemp
A$(4,3) = Noakes
A$(5,3) = Tait

SAQ 4

```
RUN
I             J              N$(I,J)

1             1              Archer
1             2              Benny
1             3              Copper
1             4              Draper
1             5              Eddie
2             1              Frame
2             2              Gwynne
2             3              Hetty
2             4              Kemp
2             5              Lamb
3             1              Morley
3             2              Noakes
3             3              Prosser
3             4              Smythe
3             5              Tait
```

(Note that Weekes was not read into the table. A 5 × 3 table will only read the first 15 items.)

Correspondence tuition

Are you studying by yourself and need help?

If you are, enrol now with NEC as a correspondence student on 30 Hour BASIC. We will give you a correspondence tutor with experience of your make of microcomputer. He will then take you through the course, marking and commenting on your assignments and giving you any advice you need on getting the programs in 30 Hour BASIC to run on your micro.

To enrol, send us the following details:

Name ....

Address ....

....

Postcode ....

Tel.No. ....

Date of birth ....

Microcomputer you are using ....

M270

Course fee:

£62 (by instalments) or £55.80 by a single payment (less £7 if you already have a copy of 30 Hour BASIC).

Also available

Cassettes

Two cassettes of the main programs in 30 Hour BASIC. Price £13.80 inc. p&p and VAT for the pair. (Available for BBC Micro and for the Spectrum).

National Extension College,
18 Brooklands Avenue, Cambridge CB2 2HN.

30 Hour BASIC ORIC edition

Flexi Study
—NEC—
RECOMMENDED TEXT ®

What is BASIC?
Microcomputers are the tool of the 80's. BASIC is the language that all of them use. So the sooner you learn BASIC, the sooner you will understand the microcomputer revolution.

30 Hour Basic is a simple self-instructional course on the language of microcomputers. But programs need more than language: they need structure as well. So the course also teaches you good programming techniques. You'll learn how to keep, order and sort files, records and directories; how to print letters and addresses; how to invent your own computer games; how to handle numbers and so on.

This edition includes a special chapter on how to use the Oric's colour, sound and graphics.

Oric edition
This is the Oric edition. A standard edition and editions for the ZX81 and Spectrum computers are also available.

£6.95 ISBN 0 86082 395 4

Cover design: Peter Hall

M270

30 Hour BASIC ORIC

BBC

NEC